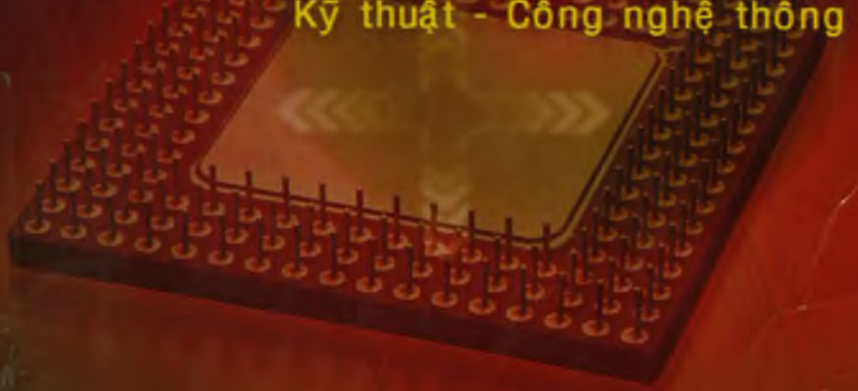
Ks. Châu Văn Trung
Cộng tác: Ts.Nguyễn Phi Khứ - Quang Hùng

GIÁO TRÌNH TIẾNG ANH
chuyên ngành
KHOA HỌC MÁY TÍNH

A Course of Basic English for Computer Science

(Dành cho sinh viên Khoa học Tự Nhiên - Kỹ thuật - Công nghệ thông tin)



NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

GIÁO TRÌNH TIẾNG ANH
CHUYÊN NGÀNH KHOA HỌC MÁY TÍNH

Ks. **Châu Văn Trung**
Cộng tác: ***Ts. Nguyễn Phi Khứ - Quang Hùng***

GIÁO TRÌNH TIẾNG ANH

Chuyên ngành

KHOA HỌC MÁY TÍNH

A course of Basic English for Computer Science

(Dành cho sinh viên Khoa học Tự nhiên - Kỹ thuật - Công nghệ thông tin)

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

Lời nói đầu

Nhằm đáp ứng nhu cầu giảng dạy và học tập của các sinh viên chuyên ngành công nghệ thông tin và kỹ thuật máy tính, chúng tôi biên soạn và xuất bản quyển "**Giáo trình tiếng Anh Chuyên ngành Khoa học Máy tính**".

Sách gồm 9 chương, trình bày những vấn đề căn bản nhất của chuyên ngành khoa học máy tính như: *các khái niệm cơ bản về máy tính,thiết kế và hoạch định chương trình, viết mã chương trình và các lệnh nhập/xuất đơn giản, cấu trúc điều khiển và vấn đề viết chương trình,các hàm và thường trình con, mảng và chuỗi, các file dữ liệu, lập trình hướng đối tượng và các cấu trúc dữ liệu.*

Bố cục mỗi chương gồm phần mục đích yêu cầu, trình bày nội dung bài học theo từng chủ điểm bằng tiếng Anh, chú thích từ vựng và hướng dẫn dịch, đọc hiểu nội dung qua tiếng Việt, sau cùng là bài tập có lời giải, bài tập bổ sung và đáp án.

Tính đa dạng và phong phú của nội dung và bố cục chặt chẽ hợp lý, dễ học khiến cho giáo trình mang tính sư phạm cao, giúp người đọc dễ dàng tiếp cận với những vấn đề mà nội dung nêu ra.

Chúng tôi hy vọng rằng giáo trình sẽ giúp ích nhiều cho các giáo viên và sinh viên trong việc tiếp cận với những vấn đề căn bản nhất của chuyên ngành khoa học máy tính.

Nhóm biên soạn

CHAPTER 1

Basic Concepts of Computers

Các khái niệm cơ bản về máy tính

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các khái niệm cơ bản về máy tính, cụ thể với các nội dung như sau:

- Computer structures
- Bus structure
- Basic operation of the computer
- Representation of data in memory
- Conversion between the binary, octal, and hexadecimal systems
- Rules for forming numbers in any system
- Arithmetic operations in the binary, octal, and hexadecimal systems
- Representing numbers in a computer

- *Cấu trúc máy tính*
- *Cấu trúc Bus*
- *Hoạt động cơ bản của máy tính*
- *Kiểu trình bày dữ liệu trong bộ nhớ*
- *Chuyển đổi giữa các hệ nhị phân, bát phân và thập lục*
- *Các quy tắc biểu diễn các số trong bất hệ thống nào*
- *Các phép toán số học trong các hệ nhị phân, thập phân và thập lục phân*
- *Trình bày các số trong máy tính*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và phần đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.

GIỚI THIỆU CHUNG

A computer is a device which, under the direction of a program, can process data, alter its own program instruction, and perform computations and logical operations without human intervention. The term **program** refers to a specific set of instructions given to the computer to accomplish a specific task. A **programmer** is a person or group of persons who write instructions to the computer. Programs, in general, are referred to as "**software**".

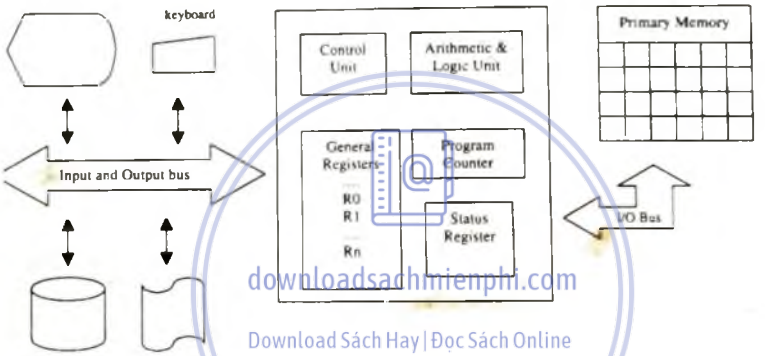
A computer can be considered at two different levels: its architecture and its implementation. The architecture consists of the user-visible interface as seen by the programmer. That is, the structure and operation of the computer from the programmer's point of view. The implementation of the computer is the construction of that interface using specific hardware (and possible software components). In this book we will refer to computer components such as monitors, printers, keyboards, and some other of its electronics as "**hardware**".

HỦ ĐIỂM 1.1

COMPUTER STRUCTURES
Cấu trúc máy tính

Most computer systems generally consist of three basic structures or ıbsystems: the high-speed memory unit, the central processing unit, and ıe peripheral devices that comprise the I/O subsystem (see Fig 1-1).

Fig. 1-1 Basic computer structure.



.1.1 The Memory Unit - *Bộ nhớ*

The memory unit of the computer, also called main memory or physical ıemory, stores all the **instructions and data** that the central processing nit can directly access and execute. The memory of majority of computers onsists of chips made of metal oxide on silicon. This type of memory is lso called Random Access ***Memory*** or RAM.

The memory of the computer is generally divided into logical units of he same size. The most common unit is called a ***byte.*** Each byte is made ıp of 8 consecutive ***bits*** or ***binary digits*** (see Fig. 1-2). Each individual bit an be magnetized to one of two different states, hence the name of binary.)ne state is said to represent 1; the other represents 0. Sometimes, these wo states are referred to as "On" and "Off" respectively.

Each byte has associated with it a unique address. According to the :onvention used, addresses can increase from right to left or left to right see Fig. 1-3). In this book we will assume that addresses increase from ·ight to left unless we specify otherwise. The *address space is* the set of all ınique addresses that a program can reference. The number of bits used to represent an address determines the size of the address space. The size of this space can be calculated by 2^N where N is the number of bits used to

represent the address. It is important to observe that there is a differenc between the address of a memory unit and the content of that unit. Th address of a byte is fixed whereas its content may vary.

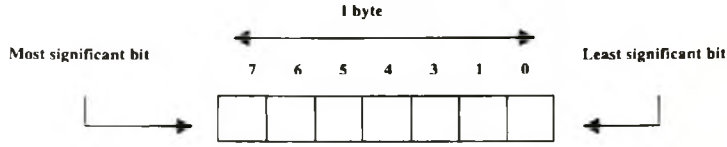


Fig. 1-2 Representation of a byte.

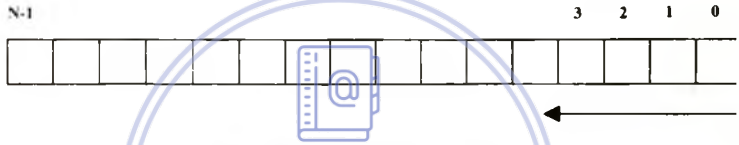


Fig. 1-3 Increasing addresses.

Bytes are also grouped into larger units. Depending on the conventic used by the manufacturers, these larger units can be called by differe names. Table 1-1 shows the common names of some of the smaller ar larger units.

Table 1-1

1 nibble	4 consecutive bits
1 byte	8 consecutive bits
1 word	2 consecutive bytes
1 longword	4 consecutive bytes
1 quadword	8 consecutive bytes
1 octaword	16 consecutive bytes

As shown in Fig. 1-2, the bits of a byte are generally numbered fro right to left beginning with 0. The rightmost bit is called the **least si nificant bit** (lsb). Likewise, the leftmost bit is called the **most signi cant bit** (msb).

Since each bit of a byte can only store one of two values, either a zero one, there are 28 differe? configurations of bits within a byte. Each cor binati)n represents a unique value. The value that each combination bits represents in a byte depends on the convention used to interpret the

values (see Example 1.5). Assuming that the bits are unsigned, the decimal value represented by the bits of a byte can be calculated as follows:

(1) Number the bits beginning on the rightmost bit position using superscripts. The superscript of the rightmost position is zero. The superscript of the next bit to its left is 1, the superscript of the next bit to the left is two and so on.

(2) Use each superscript as the exponent of a power of 2.

(3) Multiply the value of each bit by its corresponding power of 2.

(4) Add the products obtained in the previous step.

EXAMPLE 1.1 What is the decimal value of the unsigned binary configuration 11001101?

1) Number the bits beginning on the rightmost bit position using superscripts as shown below.

$$1^7 1^6 0^5 0^4 1^3 1^2 0^1 1^0$$

2) Use each superscript as the exponent of a power of 2.

$$(1*2^7) + (1*2^6) + (0*2^5) + (0*2^4) + (1*2^3) + (1*2^2) + (0*2^1) + (1*2^0)$$

3) Multiply the value of each bit by its corresponding power and add the results

$$(1*128) + (1*64) + (0*32) + (0*16) + (1*8) + (1*4) + (0*2) + (1*1) = 205$$

Therefore, the decimal value of the unsigned binary configuration 11001101 is 205. Remember that 2^0, by definition, is equal to 1. Another method to calculate the value of an unsigned binary number is explained in solved problem 1.19.

The size of the memory of a computer, as indicated before, is measured in bytes. However, the size of the memory is generally expressed in larger units. One of the most common units is the Kilobyte or Kbyte or simply K. In computer lingo, 1 K is equal to 1024 bytes. Whenever a rough approximation is required, 1K can be considered as being equivalent to 1000 bytes. Table 1-2 shows some of the other units currently used to measure primary memory. As of the writing of this book, the Megabyte or MB ($=10^6$ bytes) is the most commonly used unit to express the size of the primary memory; however, it is expected that some of the larger units will be used more frequently in the near future. The symbol should be read as "approximately equal to."

Table 1-2

1 Kilobyte	=1024 bytes
1 Megabyte	$\approx 10^6$ bytes
1 Gigabyte	$\approx 10^9$ bytes
1 Terabyte	$\approx 10^{12}$ bytes
1 Petabyte	$\approx 10^{15}$ bytes
1 Exabyte	$=10^{18}$ bytes

EXAMPLE 1.2 A computer is advertised as having a primary memory of 32 Megabytes or "Megs." What is the true size of the memory in bytes?

$$32 \text{ Mbytes} = 32*10^3 \text{ Kbytes} = 32*10^3*1024 \text{ bytes} = 32{,}768{,}000 \text{ bytes}$$

1.1.2 The Central Processing Unit - *Bộ xử lý trung tâm*

The **Central Processing Unit** (CPU) or processor is the "brain of the computer." It is in the CPU where most of the activity that occurs inside the computer takes place. The CPU is generally subdivided into two basic subunits: the **Arithmetic-Logic Unit** (ALU), and the **Control Unit** (CU).

The main activity of the CPU consists of retrieving (fetching) instructions from memory and executing these instructions. As the instructions are fetched from memory they are decoded. Decoding an instruction means interpreting what the instruction is all about and what are its operands, if any. Executing means doing what the instruction is meant to do.

The ALU or the Arithmetic-Logic Unit of the CPU, as its name implies, is where the arithmetic operations (addition, subtraction, etc.) are performed. Similarly, other operations such as the comparison of two numbers are also carried out in the ALU. Internal to the CPU there are a number of "general" registers that provide local, high-speed storage for the processor. Consider a typical situation where two numbers located in main memory are to be added. This operation may be executed as follows: first, the numbers, located in main memory, are fetched into the internal registers of the ALU where the addition is carried out. The sum, if necessary, may be stored back into some particular memory location.

In addition to these high-speed internal registers, the CPU contains one or more "status registers" that provide information about the state of the processor, the instruction being processed, any other special condition that may have occurred, and the actions that need to be taken to handle these special conditions.

The Control Unit in the CPU manages the movement of data within the processor. For example, to add the numbers of the previous example, the operands had to be moved from memory to the ALU. It is the responsibility of the control unit to manage this movement of data. If the result of the addition is to be stored back in memory the control unit will manage this task too. Incorporated within the control unit is a decoder which determines the operation that the computer needs to carry out.

1.1.3 The Input and Output Unit: *Bộ nhập và xuất*

Computers accept information via input devices. Input devices are employed by the user to send information to the computer. Two of the most common input devices are the keyboard and the mouse. Output devices are used to send information to the user. The most common output devices are the monitor and the printer. Other devices such as some of the external storage units (hard disks, tapes, jazz or zip drives) may serve a dual purpose. Input and output are among the most complex operations carried out by the CPU. Details about the physical characteristics of the I/O devices and the data format that they require are handled by system programs that are invisible to the user. A discussion of the issues involved in I/O processing is beyond the scope of this book.

CHÚ THÍCH TỪ VỰNG

- program : *chương trình*
- programmer : *người lập trình*
- software : *phần mềm*
- hardware : *phần cứng*
- instructions and data : *chỉ lệnh và dữ liệu*
- Memory : *bộ nhớ*
- bits or binary digits : *các bit và các chữ số nhị phân*
- Central Processing Unit : *bộ xử lý trung tâm*
- Arithmetic-Logic Unit : *bộ logic số học*
- Control Unit : *bộ điều khiển*

HƯỚNG DẪN ĐỌC HIỂU 1.1

Một máy tính là một thiết bị dưới sự điều khiển của một chương trình, có thể xử lý dữ liệu, tự nó biến đổi cấu trúc chương trình, và thực hiện các phép tính và các phép tính logic mà không cần sự can thiệp của con người. Thuật ngữ program chỉ ra một tập hợp các lệnh đặc biệt để đưa vào máy tính nhằm thực thi một nhiệm vụ đặc biệt.

Một programmer (một người lập trình) là một người hoặc một nhóm người viết các câu lệnh cho máy tính. Nhìn chung, các chương trình được biết như là 'software (phần mềm)'.

Một máy tính có thể được xem xét theo hai góc độ khác nhau: cấu trúc máy tính của nó và tính thực thi của nó. Cấu trúc này bao gồm giao diện người dùng hiển thị xét theo góc độ nhà lập trình. Điều này có nghĩa là cấu trúc và các phép tính toán máy tính theo quan điểm của lập trình viên. Tính thực thi của máy tính là cấu trúc của giao diện bằng cách sử dụng phần cứng (và cũng có thể là phần mềm). Trong sách này bạn sẽ tham khảo về các thành phần của máy tính chẳng hạn như màn hình, máy in, bàn phím và một số thiết bị điện tử khác được biết như là "phần cứng".

1.1 CẤU TRÚC MÁY TÍNH

Hầu hết các hệ thống máy tính bao gồm ba cấu trúc cơ bản hoặc các hệ thống phụ: bộ nhớ tốc độ cao, bộ xử lý trung tâm, và các thiết bị ngoại vi bao gồm các hệ thống phụ I/O (xem hình 1.1)

1.1.1 Bộ nhớ

Bộ nhớ của máy tính cũng được gọi là bộ nhớ chính hoặc bộ nhớ vật lý lưu trữ cấu trúc và dữ liệu để cho bộ nhớ trung tâm có thể truy cập và xử lý trực tiếp. Hầu hết bộ nhớ của máy tính có chứa các con chip được làm từ oxit kim loại trên silicon. Loại bộ nhớ này còn được gọi là bộ nhớ truy cập ngẫu nhiên hoặc RAM.

Bộ nhớ của máy tính thường được chia thành các đơn vị logic theo cùng kích cỡ. Các đơn vị phổ biến này cũng được gọi là byte. Mỗi byte được làm từ 8bit liên tục hoặc các số nhị phân (xem hình 1.2). Mỗi bit riêng lẻ có thể được tử hoa một trong hai trạng thái khác nhau theo tên nhị phân. Một trạng thái được mô tả là 1; trạng thai khác là 0. Đôi lúc, hai trạng thái này còn được gọi là "On" và "Off".

Mỗi byte liên kết với một địa chỉ duy nhất. Dựa vào tính tỷ lệ khi sử dụng, các địa chỉ này có thể tăng từ phải sang trái hoặc từ trái sang phải (xem hình 1.3). Trong sách này chúng ta sẽ giả sử rằng các địa chỉ sẽ tăng từ phải sang trái nếu không chúng ta sẽ nói về trường hợp khác. Không gian địa chỉ là tập hợp các địa chỉ để một chương trình có thể tham chiếu. Số các bit sử dụng để mô tả các địa chỉ được xác định bằng kích cỡ của không gian địa chỉ. Kích cỡ của không gian này có thể được tính bằng 2^n với n là số các bit được sử dụng để mô tả địa chỉ này. Điều này rất quan trọng trong việc quan sát sự khác biệt giữa địa chỉ bộ nhớ và nội dung của bộ nhớ đó. Địa chỉ của một byte là cố định trong khi đó nội dung của nó có thể biến đổi.

Các byte cũng có thể được gom nhóm để tạo thành các bộ lớn hơn. Tùy

thuộc vào sự thuận lợi của các nhà sản xuất, các đơn vị lớn hơn này có thể được gọi bằng nhiều tên khác nhau. Bảng 1.1 trình bày các tên phổ biến của các đơn vị nhỏ hơn và lớn hơn.

Như được minh họa trong hình 1.2 các bit của một byte thường được đánh số từ phải sang trái bắt đầu là 0. Bit bên phải nhất được gọi là least significant bit (lsb). Tương tự như vậy, bit bên trái nhất được gọi là most significant bit (msb).

Bởi vì mỗi bit của một byte chỉ có thể được lưu chỉ một trong hai giá trị, hoặc là zero hoặc là 1, do đó có 2^8 số bit khác nhau trong một byte. Mỗi một tổ hợp mô tả cho một giá trị riêng biệt. Giá trị của mỗi tổ hợp bit mô tả trong một byte phụ thuộc vào quy ước sử dụng để giải thích cho các giá trị đó (xem ví dụ 1.5). Giả sử rằng các bit này chưa được ký hiệu, thì giá trị thập phân mô tả theo các bit của một byte có thể tính toán như sau:

(1) Đánh số thứ tự cho các bit bắt đầu tại vị trí bit phải nhất theo cách đánh số mũ. Số mũ của vị trí phải nhất là zero. Số mũ của các bit kế tiếp ở bên trái là 1, số mũ của bit kế tiếp ở bên trái là hai và ....

(2) Sử dụng mỗi số ở mũ giống như số mũ của 2.

(3) Nhân giá trị của mỗi bit với lũy thừa tương ứng là 2.

(4) Cộng các tích số được cho theo các bước.

***Ví dụ 1.1** Giá trị thập phân của một số nhị phân có dạng 11001101 là bao nhiêu?*

(1) Đánh số thứ tự các bit bắt đầu tại bit phải nhất bằng cách sử dụng số mũ như được trình bày dưới đây.

$$1^7 1^6 0^5 0^4 1^3 1^2 0^1 1^0$$

(2) Mỗi số mũ là lũy thừa của 2.

$$(1*2^7) + (1*2^6) + (0*2^5) + (0*2^4) + (1*2^3) + (1*2^2) + (0*2^1) + (1*2^0)$$

(3) Nhân giá trị của mỗi bit với số mũ tương ứng rồi cộng kết quả lại.

$$(1*128) + (1*64) + (0*32) + (0*16) + (1*8) + (1*4) + (0*2) + (1*1) = 205$$

Do đó giá trị thập phân của một số nhị phân 11001101 là 205. Hãy nhớ rằng 2^0 theo định nghĩa là 1. Các phương pháp để tính giá trị của một số nhị phân được giải thích trong bài thực hành 1.19.

Kích thước bộ nhớ của máy tính như được trình bày trước đây được đo bằng byte. Tuy nhiên kích thước bộ nhớ máy tính được mô tả ở đơn vị lớn hơn. Một trong những đơn vị thường dùng là kilobyte hoặc Kbyte

hoặc đơn giản là K. Theo ngôn ngữ máy tính, 1 K = 1024 byte. Tuy nhiên, khi tính gần đúng, một ký có thể được lên gần bằng 1000 byte. Bảng 1.2 trình bày các đơn vị hiện hành hiện đang sử dụng để tính toán bộ nhớ chính. Trong sách này, Megabyte hoặc MB ($\approx 10^6$ bytes) là đơn vị tính toán thông dụng nhất để trình bày kích cỡ bộ nhớ chính; tuy nhiên, trong tương lai gần đây các đơn vị lớn hơn sẽ được dùng thường xuyên hơn. Ký hiệu $\approx$ sẽ được đọc là "xấp xỉ bằng".

***Ví dụ 1.2** Một máy tính được quảng cáo có bộ nhớ chính là 32 megabytes gọi "Megs". Vậy kích cỡ bộ nhớ là bao nhiêu byte?*

32 Mbytes = $32*10^3$ Kbytes = $32*10^3*1024$ bytes = 32,768,000 bytes

1.1.2 Bộ xử lý trung tâm

Bộ xử lý trung tâm (CPU) hoặc trình xử lý là "bộ não của máy tính". Nó nằm trong CPU nơi mà hầu hết các hoạt động diễn ra trong máy tính đều xảy ra. CPU thường được chia thành hai đơn vị con cơ bản: Đơn vị số học [Arithmetic-Logic Unit (ALU)], và Bộ điều khiển (Control Unit - CU).

Hoạt động chính của CPU bao gồm các lệnh truy hồi (gọi lại) từ bộ nhớ và chạy các lệnh này. Khi các lệnh này được gọi lại từ bộ nhớ và chúng sẽ được giải mã. Việc giải mã một câu lệnh có nghĩa là giải mã ít câu lệnh này và tất cả những toán tử của nó nếu có. Việc xử lý có nghĩa là thực thi những gì câu lệnh đó yêu cầu.

ALU hoặc Arithmetic-Logic Unit (Đơn vị số học logic) của CPU,là nơi mà các phép toán số học (cộng, trừ, ...) được thực thi. Tương tự, các phép toán khác chẳng hạn như so sánh giữa hai số cũng được tiến hành trong ALU. Bên trong CPU có các thanh ghi tổng quát để cung cấp vị trí, vùng lưu trữ tốc độ cao , bộ xử lý. Giả sử trong tình huống này có hai số nằm tại bộ nhớ chính được thêm vào. Phép tính này có thể được thực hiện như sau; đầu tiên, các số nằm ở bộ nhớ chính sẽ được gọi vào các thanh ghi bên trong của ALU nơi mà phép cộng được thực hiện. Nếu cần, tổng có thể được lưu trữ trở lại trong vùng nhớ đặc biệt.

Ngoài những thanh ghi bên trong có tốc độ cao này, thì CPU có chứa một hoặc nhiều "thanh ghi trạng thái' để cung cấp thông tin về trạng thái của bộ xử lý này, lệnh đang được xử lý và bất kỳ các điều kiện đặc biệt nào có thể xảy ra khác, và các hành động cần thiết để tác động theo một các điều kiện đặc biệt này.

Bộ điều khiển trong CPU quản lý sự di chuyển dữ liệu bên trong bộ xử lý. Ví dụ để cộng thêm các số của ví dụ trước, các toán hạng phải được di chuyển từ bộ nhớ sang ALM. Trách nhiệm của bộ điều khiển là quản lý sự di chuyển của dữ liệu. Nếu kết quả của phép cộng được lưu trữ lại bên trong bộ nhớ thì bộ điều khiển này cũng sẽ quản lý

nhiệm vụ này. Sự kết hợp bên trong bộ điều khiển là một bộ giải mã để xác định phép toán mà máy tính cần phải thực hiện.

1.1.3 Bộ nhập và xuất

Máy tính đảm nhận thông tin thông qua các thiết bị nhập. Các thiết bị nhập được người dùng sử dụng để gởi thông tin vào máy tính. Hai thiết bị nhập phổ biến nhất đó là bàn phím và chuột. Các thiết xuất thường được sử dụng để gởi thông tin cho người dùng. Các thiết bị xuất phổ biến đó là màn hình và máy in. Các thiết bị khác chẳng hạn như các bộ lưu trữ ngoài (các đĩa cứng, các băng từ, các đĩa jazz hoặc zip) có thể phục vụ cho hai mục đích. Nhập và xuất là một trong các phép tính phức tạp nhất mà CPU phải thực hiện. Các chi tiết về các ký tự vật lý của các thiết bị cho nhập xuất (I/O) và định dạng dữ liệu mà chúng yêu cầu được xử lý bởi hệ thống chương trình vốn không hiển thị cho người dùng. Vấn đề xử lý nhập xuất (I/O) được thảo luận sau chương này.

CHỦ ĐIỂM 1.2

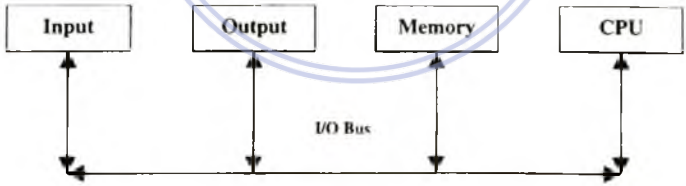
BUS STRUCTURE
Cấu trúc Bus

To transfer data within its different components the computer uses a collection of wires called a bus. A computer system typically has three such buses: the address bus, the data bus, and the control bus. The data bus is used for transmission of data. A data bus must have as many wires as there are bits in the memory unit selected for a particular computer. To access data in memory it is necessary to issue an address to indicate its location. The CPU sends address bits to memory via the address bus.

Figure 1-4 shows a two-bus structure computer. The CPU interacts with memory via the memory bus. Input and Output functions are conducted via the I/O bus. Other configurations are also used. Figure 1-5 shows a single-bus structure used in some small computers.

Fig. 1-4 A two-bus structure.

Fig. 1-5 Single bus structure.



EXAMPLE 1.3 Assuming that the basic unit of a computer is a word, how many bits can be transferred by the data bus at any one time?

The answer depends on the number of bits that are in a word. Using Tablel-1 the number of bits that needs to be transferred is at least 16 bits. We say at least since some additional control information may be carried on the data bus.

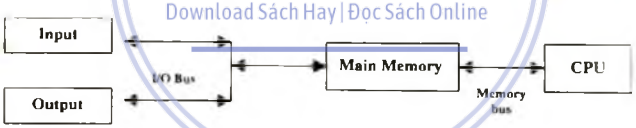
CHÚ THÍCH TỪ VỰNG

- address bus : *bus địa chỉ*
- data bus : *bus dữ liệu*
- control bus : *bus điều khiển*
- Input and Output functions : *các chức năng nhập và xuất*

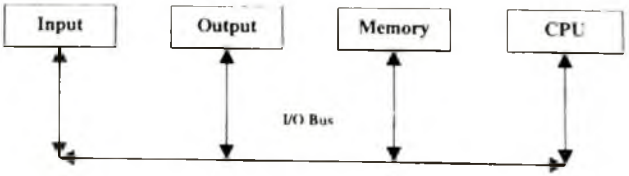
HƯỚNG DẪN ĐỌC HIỂU 1.2

Để vận chuyển dữ liệu bên trong các thành phần khác nhau, máy tính sử dụng một tập hợp các dây được gọi là bus. Một hệ thống máy tính thường có ba bus: bus địa chỉ, bus dữ liệu và bus điều khiển. Bus dữ liệu được dùng để chuyển tải dữ liệu. Một bus dữ liệu phải có nhiều đầu khi có nhiều bit trong bộ nhớ được chọn cho một máy tính đặc biệt. Để truy cập dữ liệu trong bộ nhớ nếu cần phải cấp phát một địa chỉ chỉ ra vị trí của nó. CPU này gởi các bit địa chỉ đến bộ nhớ thông qua bus địa chỉ.

Hình 1.4 hiển thị một cấu trúc hai bus của máy tính. CPU này tương tác với bộ nhớ thông qua bus bộ nhớ này. Các chức năng nhập và xuất được hướng dẫn thông qua bus I/O. Các cấu hình khác cũng được sử dụng. Hình 1.5 minh hoạ một cấu trúc bus đơn được dùng trong nhiều máy tính nhỏ.



***Hình 1.4** Cấu trúc hai bus*



***Hình 1.5** Cấu trúc của bus đơn*

***Ví dụ 1.3** Giả sử rằng đơn vị cơ bản của một máy tính là từ, có bao nhiêu bit có thể chuyển tải bởi bus dữ liệu cùng một lúc?*

Câu trả lời dựa trên số bit trong một từ. Sử dụng bảng 1.1 số các bit cần để chuyển tải tạo một thời điểm ít nhất là 16 bit. Ở đây, chúng ta nói ít nhất bởi vì sẽ có một số thông tin điều khiển có thể được mang theo bus dữ liệu.

CHỦ ĐIỂM 1.3

BASIC OPERATION OF THE COMPUTER
Hoạt động cơ bản của máy tính

As previously indicated, the operation of the computer is controlled by the instructions of a program. The CPU fetches the individual instructions directly from memory before executing them; however, before executing the instructions, how does the computer know what each instruction means? The answer to this question is easy. The computer decodes each instruction according to a predefined format. The general format of an instruction may look like this:

operator [operand1], [operand2], [operand3]

where the operator indicates the type of action to be performed (ADD, SUBTRACT, MULTIPLY, etc.) and the operands are the "pieces" of information on which the action indicated by the operator is going to be performed. The square brackets around the operands indicate that the operands are optional. According to this general format, an instruction may have zero, one, two, or three operands. The reader should be aware that the computer only sees a sequence of 0's and 1's. It does not see an operator like "ADD" or "SUBTRACT"; instead it sees their predefined equivalent numerical code such as 10000001 for ADD or 10000110 for MULTIPLY.

The format of a typical instruction may look something like this:

ADDW3 location1, location2, result

This particular instruction adds the contents of the words at the addresses called location1 and location2 and stores the sum at the address called result. The "W" indicates that the operands are words whereas the suffix "3" indicates that there are three operands in this instruction.

This instruction requires several steps to be carried out. First, the instruction is transferred from memory into the CPU. The location of the instruction currently being executed is kept in one of the CPU special registers called the **instruction counter (IC) or program counter** (PC)-the terminology varies with the manufacturer. The instruction just fetched is stored in another special register called the **Instruction Register** (IR). The control unit decodes the instruction and recognizes it as an ADD operation with three word operands. Then, the first operand (location1) and second operand (location2) are fetched into two general registers. These two registers are added and the result stored back into memory in the location called result. After the two operands have been fetched the program counter is updated so it points to the next instruction to be executed. During this process, two additional registers facilitate the communication between the CPU and main memory. These registers are the memory ***ad-***

dress register (MAR) and the **memory *data*** register (MDR). The MAR is used to hold the address of the location to or from which data are being transferred. The MDR contains the data to be written into or read out of the addressed location.

EXAMPLE 1.4 Assume that a computer has two instructions for addition with the following formats: ADDW2 locationl, location2 and ADDW3 locationl, location2, result. Does it make any difference to use one instruction over the other?

To answer this question properly we need to understand the differences between these two instruction formats. The first ADD instruction contains two word operands. Hence, the name ADDW2. Let us assume that in this type of instruction the result of the operation, as defined by the computer manufacturer, is stored into the second operand. This type of ADD operation is called a destructive add instruction since the value of location2 will be replaced by the result of the addition (see Fig. 1-6).

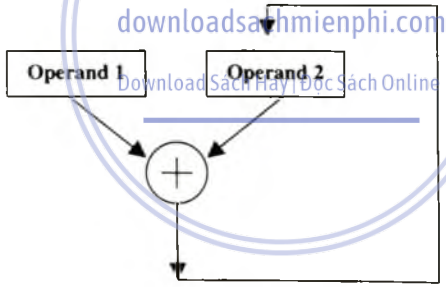


Fig. 1-6 Destructive add.

The second instruction contains three word operands, hence the name ADDW3. In this type of ADD instruction the result of the addition of the first two operands is stored into the location specified by the third operand. This type of instruction is called a non-destructive add since the values of the operands remain intact after the instruction is executed (see Fig. 1-7).

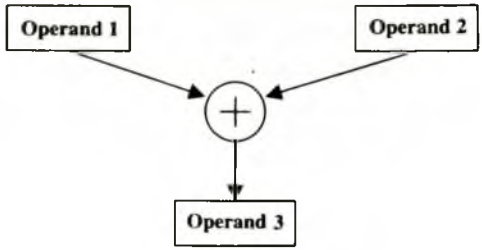


Fig. 1-7 Non-destructive add.

CHÚ THÍCH TỪ VỰNG

- instruction counter or program counter : *bộ đếm chương trình*
- Instruction Register : *thanh ghi lệnh*
- memory address register : *thanh ghi địa chỉ bộ nhớ*
- memory data register : *thanh ghi dữ liệu bộ nhớ*

HƯỚNG DẪN ĐỌC HIỂU 1.3

Như đã học ở phần trước, phép toán của máy tính được điều khiển bởi các lệnh của chương trình. CPU gọi các lệnh đơn một cách trực tiếp từ bộ nhớ trước khi thực thi nó; tuy nhiên, trước khi thực thi câu lệnh, làm sao máy tính có thể biết được ý nghĩa của mỗi câu lệnh? Câu trả lời thật đơn giản. Máy tính giải mã mỗi câu lệnh dựa trên các dạng thức được định nghĩa sẵn. Dạng của câu lệnh có thể giống như sau:

toán tử [toán hạng1], [toán hạng2], [toán hạng3]

ở đó các toán tử chỉ ra loại hành động để thực hiện các phép tính (cộng, trừ, nhân,....) và các toán hạng là một phần của thông tin của câu lệnh trên hành động được chỉ ra bởi toán tử để được thực hiện. Các dấu ngoặc vuông xung quanh các toán hạng chỉ ra các toán hạng này là tùy ý. Dựa trên dạng thức tổng quát này, một câu lệnh có thể có zero, một, hai, hoặc ba toán hạng.

Người đọc lưu ý rằng máy tính chỉ hiểu được chuỗi 0 và 1. Nếu không hiểu một toán hạng chẳng hạn như "ADD" hoặc "SUBTRACT"; thay vì nó chỉ hiểu các mã tương đương được định nghĩa sẵn chẳng hạn như 10000001 cho ADD (phép cộng) hoặc 10000110 cho MULTIPLY (phép nhân).

Dạng thức của một lệnh thông thường có thể như sau:

ADDW3 loaction1, loaction2, result

Lệnh đặc biệt này cộng nội dung của các từ tại địa chỉ được gọi là location 1 và location 2 và lưu tổng này tại địa chỉ gọi là kết quả. Từ "W" chỉ ra các toán hạng là từ trong khi hậu tố "3" chỉ ra có ba toán tử trong câu lệnh này.

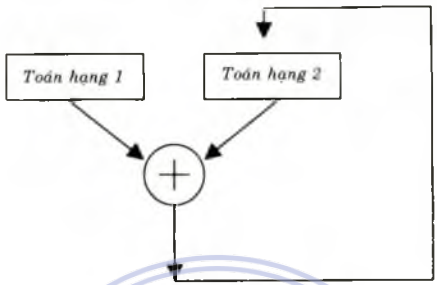
Câu lệnh này yêu cầu nhiều bước thực thi. Trước tiên, câu lệnh này được chuyển từ bộ nhớ vào CPU. Vị trí của câu lệnh hiện đang được thực thi sẽ được giữ tại một trong những thanh ghi đặc biệt của CPU được gọi là instruction counter (IC) (bộ đếm lệnh) hoặc program counter (PC) (bộ đếm chương trình) - thuật ngữ này khác nhau tùy theo nhà sản xuất. Câu lệnh vừa mới được gọi sẽ được lưu trữ tại một thanh ghi đặc biệt khác được gọi là Instruction Register (IR) (thanh ghi lệnh). Bộ điều khiển giải mã lệnh này và nhận ra nó là ADD (phép cộng) với ba toán tử. Sau đó, toán hạng đầu tiên (loaction 1) và toán hạng thứ hai là (location 2) sẽ được gọi vào hai thanh ghi tổng quát. Hai thanh ghi này được cộng và kết quả được lưu trở lại bộ nhớ tại vị trí gọi là kết quả. Sau khi hai toán hạng này được gọi thì bộ đếm chương trình được cập nhật để câu lệnh tiếp theo được thực thi. Trong suốt quá trình xử lý này, hai thanh ghi cộng này làm cho việc giao tiếp giữa CPU trở nên dễ dàng hơn. Những thanh ghi này là thanh ghi địa chỉ có memory address register (MAR) (thanh ghi địa chỉ bộ nhớ) và memory data register (MDR) (thanh ghi bộ nhớ dữ liệu). MAR dùng để giữ địa chỉ của vùng xuất phát hoặc đi từ dữ liệu đang được vận chuyển. MDR có chứa dư 4liệu được viết vào hoặc đọc ra vị trí địa chỉ.

***VÍ DỤ 4** Giả sử rằng máy tính có hai lệnh để thực thi phép cộng với dạng thức sau: ADDW2 loaction1, location2 và ADDW3 loaction1, location2, result. Có sự khác biệt nào sử dụng một trong các chỉ lệnh này?*

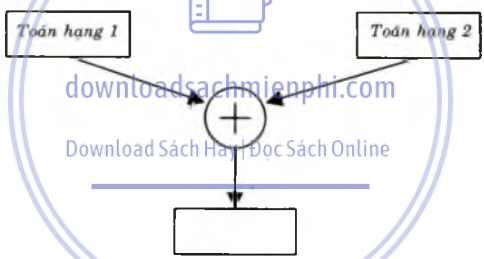
Để trả lời câu hỏi này chính xác thì chúng ta cần phải hiểu sự khác biệt giữa hai câu lệnh. Trước tiên câu lệnh ADD có chứa hai toán hạng. Ở đây, tên là ADDW2. Chúng ta giả thiết rằng trong câu lệnh này kết quả của phép toán, được xác định bởi máy tính sẽ được lưu trữ trong toán hạng thứ hai. Loại phép tính ADD (cộng) này được gọi là một phép tính cộng tiêu cực bởi vì giá trị tại location2 được thay thế bằng kết quả của phép cộng (xem hình 1.6).

Câu lệnh thứ hai có chứa ba toán hạng. Ở đây có tên là ADDW3. Trong loại câu lệnh ADD này kết quả của phép cộng giữa hai toán tử sẽ được lưu trữ trong vị trí được chỉ định ở toán hạng thứ ba. Loại câu lệnh này được gọi là non-destructive add (cộng không phá hủy cấu

trúc) bởi vì các giá trị của các toán hạng vẫn được giữ nguyên sau khi lệnh này được thực thi (xem hình 1.7)



Hình 1.6 Cộng tiêu cực



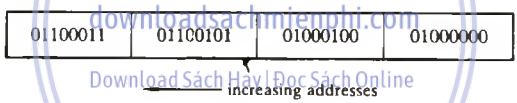
Hình 1.7 Cộng tích cực

CHỦ ĐIỂM 1.4

REPRESENTATION OF DATA IN MEMORY
Kiểu trình bày dữ liệu trong bộ nhớ

If we could look at the content of the memory of a computer we would find that all types of data and instructions are represented using only 0's and 1's. Therefore, an obvious question is how can the computer tell the type of information that is stored on a particular location? The answer is simple; the computer knows the type of data stored in a particular location from the context in which the data are being used. This allows the computer to interpret it properly. Example 1.5 shows that a string of data may have different interpretations.

EXAMPLE 1.5 Given the sequence of 4 bytes shown below, what would be the values of this sequence if we consider it as being formed by (a) four individual bytes? (b) two-word integers with two bytes per word? (c) a longword integer with four bytes per longword?



The value of each individual byte can be calculated using the procedure indicated in Section 1.1.1. The subscript indicated in parenthesis at the right-hand corner of a number indicates if the number is binary $_{(2}$ or decimal $_{(10}$. This number, as we will see later, is called the base or radix of the number.

(a) Considering the bytes from right to left, we have

$$01000000_{(2} = 0^7 1^6 0^5 0^4 0^3 0^2 0^1 0^0 = (1*2^6) = 64_{(10}$$
$$01000100_{(2} = 0^7 1^6 0^5 0^4 0^3 1^2 0^1 0^0 = (1*2^6) + (1*2^2) = 64 + 4 = 68_{(10}$$
$$01100101_{(2} = 0^7 1^6 1^5 0^4 0^3 1^2 0^1 1^0 = (1*2^6) + (1*2^5) + (1*2^2) + (1*2^0) = 64 + 32 + 4 + 1 = 101_{(10}$$
$$01100011_{(2} = 0^7 1^6 1^5 0^4 0^3 0^2 1^1 1^0 = (1*2^6) + (1*2^5) + (1*2^1) + (1*2^0) = 64 + 32 + 2 + 1 = 99_{(10}$$

(b) The first two-byte word consists of the two rightmost bytes; using the procedure of Section 1.1.1 we have that

$$0100010001000000_{(2} = 0^{15} 1^{14} 0^{13} 0^{12} 0^{11} 1^{10} 0^9 0^8 0^7 1^6 0^5 0^4 0^3 0^2 0^1 0^0$$
$$= (1*2^{14}) + (1*2^{10}) + (1*2^6)$$
$$= 17,472_{(10}$$

The next two-byte word is 0110001101100101. Using the procedure of Section 1.2.1, we have that

$$0110000110110010 1_{(2} = 0^{15}1^{14}1^{13}0^{12}0^{11}0^{10}1^{9}1^{8}0^{7}1^{6}1^{5}0^{4}0^{3}1^{2}0^{1}1^{0}$$
$$= (1*2^{14}) + (1*2^{13}) + (1*2^{9}) + (1*2^{8}) + (1*2^{6}) + (1*2^{5}) + (1*2^{2}) + (1*2^{0})$$
$$= 24,445$$

(c) Using the procedure of Section 1.2.1 to calculate the value of the longword consisting of four bytes we have that

$$0^{31}1^{30}1^{29}0^{28}0^{27}0^{26}1^{25}1^{24}0^{23}1^{22}1^{21}0^{20}0^{19}1^{18}0^{17}1^{16}0^{15}1^{14}0^{13}0^{12}0^{11}1^{10}0^{9}0^{8}0^{7}1^{6}0^{5}0^{4}0^{3}0^{2}0^{1}0^{0}{}_{(2} =$$
$$(1*2^{30}) + (1*2^{29}) + (1*2^{25}) + (1*2^{24}) + (1*2^{22}) + (1*2^{21}) + (1*2^{18}) + (1*2^{16}) + (1*2^{14}) +$$
$$(1*2^{10}) + (1*2^{6}) = 1,667,580,992_{(10}$$

1.4.1 Number Systems and Codes - *Hệ thống số và mã*

The most commonly used numbering system is the decimal system. However, due to the binary nature of its electronic devices, the use of the decimal system by the computer is limited. Most of the data representation and arithmetic operations are carried out in the binary system or some of its "shorthand" notation such as the octal or hexadecimal systems. However, before considering the binary, octal, or hexadecimal systems, let us review some of the characteristics of the decimal system.

The decimal system, as its name indicates, is based on the number 10. The number 10 itself is called the **basis or radix** of the system. In any numerical system, the basis tells us how many different symbols there are in the system. In the decimal system these symbols are called digits. They are the familiar 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Notice that they range in value from 0 to 9. This is a particular case of a more general rule that can be stated as follows:

"Given any positive integer basis or radix N, there are N different individual symbols that can be used to write numbers in the system. The value of these symbols ranges from 0 to N - 1."

In addition to the binary system, the computer and telecommunication fields make extensive use of some other numerical systems. They are the octal (its basis is 8) and the hexadecimal (its basis is 16) systems. Example 1.6 explains the basic elements of these systems.

EXAMPLE 1.6 How many different symbols are there in the octal and hexadecimal systems? What are the ranges of these symbols?

The basis of the octal system is 8. That is, N = 8. Therefore, there are 8 different symbols. The range of these values varies from 0 to (8 - 1). That is, from 0 to 7. These symbols are: 0, 1, 2, 3, 4, 5, 6, and 7. We call all of these symbols by their decimal names: one, two, three, and so on.

The basis of the hexadecimal system is 16. That is, N = 16. Therefore, there are 16 different symbols. The values of these symbols range from 0

to (16 -1) or from 0 to 15. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Notice that after 9, the symbols are the letters A through F. In this system, the letter A stands for 10, the B for 11, the C for 12, the D for 13, the E for 14, and the F for 15. The reason for choosing letters instead of combination of numerical symbols to represent symbols higher than 9 is to keep all symbols single characters.

To indicate the basis on which a given number is written, we will use a subscript at the lower right side of the number. As indicated in Section 1.4, the notation $0101_{(2}$ indicates that the number is binary. Likewise, the notation $01256_{(8}$ will indicate that the number is written in base 8. When referring to numbers of the decimal system, the subscript is generally omitted. However, we will use it whenever it is necessary to clarify the text. Although the word digit generally refers to the individual symbols of the decimal system, it is also common to call the individual symbols of the other numerical systems by this name.

1.4.2 Positional Systems - *Vị trí trong các hệ*

All the numerical systems that we have considered so far, including the decimal system, are **positional** systems. That is, the value represented by a symbol in the numerical representation of a number depends on its position in the number. For example, in the decimal system, the 4 in the number 478 represents 400 units; the 4 in the number 547 represents 40 units. This fact can be seen more clearly if we decompose the number as follows:

$$478 = \mathbf{400} + 70 + 8$$
$$547 = 500 + \mathbf{40} + 7$$

In any positional system, the value of any digit in the representation of a number can be calculated using the following steps: (1) Number the digits from right to left using superscripts. Begin with zero, as the superscript of the rightmost digit, and increase the superscripts by 1 as you move from left to right. (2) Use each superscript to form a power of the basis. (3) Multiply the digit's own value, in decimal, by its corresponding power of the basis.

To calculate the decimal equivalent of the entire number, sum all the products obtained in step 2.

EXAMPLE 1.7 What is the value of each digit in the number $1228_{(10}$?

The basis is 10 since the number is a decimal number. To calculate the value of each digit follow the steps indicated below.

(1) Number the digits using superscripts. Begin with 0 in the rightmost

position and increase the superscripts as you move from right to left. The number should look like this:

$$1^3 2^2 2^1 8^0$$

(2-3) Use the superscripts to form powers of the basis and multiply the digit's own value, in decimal, by its corresponding power:

The value of 1 in 1228 is equal to $1*10^3 = 1*1000 = 1000$.

The value of the first 2 (from left to right) in 1228 is equal to $2*10^2 = 2*100 = 200$

The value of the second 2 (from left to right) in 1228 is equal to $2*10^1 = 2*10 = 20$

The value of 8 in 1228 is equal to $8*10^0 = 8*1 = 8$.

EXAMPLE 1.8 What is the value of each digit in the number $1253_{(8}$?What is the decimal equivalent of the number?

The basis is 8 since the number is written in the octal system. Using a procedure similar to the one in the previous example, we can use superscripts and write the number as $1^3 2^2 5^1 3^0$. Using these superscripts as the power of the basis we have:

The value of 1 in 1253 is equal to $1*8^3 = 1*512 = 512$

The value of 2 in 1253 is equal to $2*8^2 = 2*64 = 128$

The value of 5 in 1253 is equal to $5*8^1 = 5*8 = 40$

The value of 3 in 1253 is equal to $3*8^0 = 3*1 = 3$

The decimal equivalent of the number is obtained by adding all the previous products.

$$\begin{aligned} 1253_{(8} &= (1*8^3) + (2*8^2) + (5*8^1) + (3*8^0) \\ &= (1*512) + (2*64) + (5*8) + (3*1) \\ &= 512 + 128 + 40 + 3 \\ &= 683 \end{aligned}$$

Notice that whenever we find the decimal equivalent of number, we also determine the individual value of each digit. We will use this combined method from now on.

EXAMPLE 1.9 What is the value of each digit in the number $1A2F_{(16}$?What is the decimal equivalent of the number?

The basis is 16 since the number is written in the hexadecimal system. Using superscripts we can write the number as $1^3 A^2 5^1 F^0$. To calculate the

decimal equivalent of this hexadecimal number, substitute the symbols A and F by their respective decimal equivalents of 10 and 15. The decimal equivalent of the number is obtained by adding all the previous powers.

$$\begin{aligned} 1A5F_{(16} &= (1*16^3) + (10*16^2) + (5*16^1) + (15*16^0) \\ &= (1*4096) + (10*256) + (5*16) + (15*1) \\ &= 4096 + 2560 + 80 + 15 \\ &= 6751 \end{aligned}$$

The value of 1 in 1A5F is equal to $1*16^3 = 1*4096 = 4096$
The value of A in 1A5F is equal to $10*16^2 = 2*256 = 2560$
The value of 5 in 1A5F is equal to $5*16^1 = 5*8 = 80$
The value of F in 1A5F is equal to $15*16^0 = 3*1 = 15$

CHÚ THÍCH TỪ VỰNG

- interpretations : *sự giải mã*
- positional systems : *vị trí trong các hệ*
- binary system : *hệ nhị phân*
- string of data : *chuỗi dữ liệu*

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 1.4

Nếu chúng ta nhìn vào nội dung bộ nhớ của một máy tính thì chúng ta sẽ biết rằng tất cả các loại dữ liệu và các loại lệnh sẽ được mô tả dưới dạng 0 và 1. Vì vậy, một câu hỏi hiển nhiên là làm cách nào máy tính hiểu được loại thông tin được lưu trữ trong vị trí đặc biệt này? Câu trả lời rất đơn giản, máy tính biết được loại dữ liệu được lưu trữ trong vùng đặc biệt này dựa theo nội dung mà dữ liệu đang được dùng. Điều này cho phép máy tính giải mã nó một cách chính xác. Ví dụ 1.5 minh họa chuỗi dữ liệu có thể có nhiều cách giải mã khác nhau.

Ví dụ 1.5 *Cho một chuỗi dữ liệu 4 byte được minh họa dưới đây, giá trị của chuỗi này là bằng bao nhiêu nếu chúng ta xem nó ở các dạng (a) bốn byte riêng biệt? (b) hai từ nguyên với hai byte trên một từ? (c) một số nguyên longword (từ dài) với bốn byte trên longword đó?*

01100011	01100101	01000100	01000000

Các địa chỉ tăng dần

Giá trị của mỗi byte có thể được tính toán bằng cách sử dụng phương pháp được hướng dẫn trong phần 1.1.1. Số mũ được chỉ trong dấu ngoặc đơn ở góc phải dưới của con số chỉ ra rằng số đó có phải là số nhị phân $_{(2}$ hoặc số thập phân $_{(10}$. Con số này chúng ta sẽ xem ở chương sau, được gọi là số cơ sở hoặc cơ số của một số.

(a) Xét các byte từ phải sang trái, chúng ta có

$01000000_{(2} = 0^7 1^6 0^5 0^4 0^3 0^2 0^1 0^0 = (1*2^6) = 64_{(10}$
$01000100_{(2} = 0^7 1^6 0^5 0^4 0^3 1^2 0^1 0^0 = (1*2^6) + (1*2^2) = 64 + 4 = 68_{(10}$
$01100101_{(2} = 0^7 1^6 1^5 0^4 0^3 1^2 0^1 1^0 = (1*2^6) + (1*2^5) + (1*2^2) + (1*2^0) = 64 + 32 + 4 + 1 = 101_{(10}$
$01100011_{(2} = 0^7 1^6 1^5 0^4 0^3 0^2 1^1 1^0 = (1*2^6) + (1*2^5) + (1*2^1) + (1*2^0) = 64 + 32 + 2 + 1 = 99_{(10}$

(b) Từ hai byte đầu tiên có chứa hai byte ở bên phải nhất; bằng các sử dụng phương pháp ở phần 1.1.1, chúng ta có

$0100010001000000_{(2} = 0^{15} 1^{14} 0^{13} 0^{12} 0^{11} 1^{10} 0^9 0^8 0^7 1^6 0^5 0^4 0^3 0^2 0^1 0^0$
$= (1*2^{14}) + (1*2^{10}) + (1*2^6)$
$= 17,472_{(10}$

Từ hai byte kế tiếp là 0110001101100101. Bằng cách sử dụng phương pháp ở phần 1.2.1, chúng ta có

$0110001101100101_{(2} = 0^{15} 1^{14} 1^{13} 0^{12} 0^{11} 0^{10} 1^9 1^8 0^7 1^6 1^5 0^4 0^3 1^2 0^1 1^0$
$= (1*2^{14}) + (1*2^{13}) + (1*2^9) + (1*2^8) + (1*2^6) + (1*2^5) + (1*2^2) + (1*2^0)$
$= 24,445$

(c) Bằng cách sử dụng phương pháp ở phần 1.2.2 để tính giá trị của của số longword có chứa bốn byte, ta có

$0^{31} 1^{30} 1^{29} 0^{28} 0^{27} 0^{26} 1^{25} 1^{24} 0^{23} 1^{22} 1^{21} 0^{20} 0^{19} 1^{18} 0^{17} 1^{16} 0^{15} 1^{14} 0^{13} 0^{12} 0^{11} 1^{10} 0^9 0^8 0^7 1^6 0^5 0^4 0^3 0^2 0^1 0^0{}_{(2} =$
$(1*2^{30}) + (1*2^{29}) + (1*2^{25}) + (1*2^{24}) + (1*2^{22}) + (1*2^{21}) + (1*2^{18}) + (1*2^{16}) + (1*2^{14}) +$
$(1*2^{10}) + (1*2^6) = 1,667,580,992_{(10}$

1.4.1 Các hệ số và mã số

Hệ số thường dùng nhất là hệ thập phân. Tuy nhiên, dựa vào trạng thái nhị phân của các thiết bị điện tử, thì cách sử dụng hệ thập phân bằng máy tính là hạn chế. Hầu hết dữ liệu được trình bày và các phép tính số học được xử lý theo hệ nhị phân hoặc một số ký hiệu tính toán của nó chẳng hạn như hệ tám (bát phân) hoặc hệ mười sáu (thập lục phân) . Tuy nhiên, trước khi xem xét các hệ nhị phân, hệ bát phân và hệ thập lục phân, chúng ta hãy xem lại các số trong hệ thập phân.

Hệ thập phân, như tên nó chỉ ra là dựa trên số 10. Số 10 này được gọi là hệ cơ sở hoặc cơ số. Trong bất kỳ hệ số nào, cơ sở này cho chúng ta biết có bao nhiêu ký tự khác nhau trong một hệ. Trong hệ thập phân

những ký hiệu này được gọi là số. Chúng là 0, 1, 2, 3, 4, 5, 6, 7, 8 và 9. Lưu ý rằng chúng nằm trong dãy giá trị từ 0 đến 9. Đây là một trường hợp phổ biến của nguyên tắc chung có thể được phát biểu như sau:

> ***"Cho bất kỳ số nguyên dương nào có cơ số N, với N là các ký tự phân biệt khác nhau mà có thể được dùng để viết các số trong hệ thống này. Giá trị của các ký tự này nằm trong khoảng từ 0 cho đến N-1."***

Ngoài hệ nhị phân, máy tính và các lĩnh vực viễn thông đều sử dụng các hệ số khác. Chúng là hệ bát phân (cơ số 8) và hệ thập lục phân (cơ số 16). Ví dụ 1.6 sẽ giải thích các thành phần cơ bản của hệ này.

***Ví dụ 1.6** Có bao nhiêu ký tự khác nhau trong hệ bát phân và hệ thập lục phân? Dãy của lý tự này là bao nhiêu?*

Cơ số của hệ bát phân là 8. Nghĩa là N = 8. Do đó, có 8 ký tự khác nhau. Dãy của ký tự này biến thiên từ 0 đến (8-1). Nghĩa là từ 0 cho đến 7. Những ký tự này là : 0, 1, 2, 3, 4, 5, 6, và 7. Chúng ta gọi những ký tự này theo tên thập phân: một, hai, ba ....

Cơ số của hệ thập lục phân là 16. Nghĩa là N = 16. Vì vậy, có 16 ký tự khác nhau. Các giá trị của dãy ký tự này là từ 0 đến (16 - 1) hoặc từ 0 cho đến 15. Những ký tự này là 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D. E và F. Lưu ý rằng sau số 9, các ký tự bắt đầu từ mẫu tự A cho đến F. Trong hệ này, mẫu tự A đại diện cho số 10, B cho 11, C cho 12, D cho 13, E cho 14 và F cho 15. Lý do chọn các ký tự thay vì kết hợp các số ký tự để diễn tả các số lớn hơn 9 sẽ là các ký tự đơn.

Để mô tả cơ số của một số đã cho , chúng ta sử dụng số mũ ở góc phải dưới bên cạnh số đó. Như được chỉ định trong phần 1.4 ký hiệu 0101_2 chỉ ra rằng số đó là hệ nhị phân. Tương tự như vậy, ký hiệu 01256_8 sẽ chỉ ra số đó được viết theo cơ số 8. Khi chuyển đến hệ thập phân, các số mũ thường không hiển thị. Tuy nhiên, chúng ta sẽ sử dụng nó nếu cần để làm cho text được dễ hiểu. Mặc dù, chữ số từ thường được tham chiếu đến các ký hiệu riêng biệt của hệ thập phân. Các hệ thống số khác cũng thường gọi tên này.

1.4.2 Vị trí trong các hệ

Tất cả các hệ số mà chúng ta xem xét kể cả hệ thập phân là vị trí các hệ. Đó là ký hiệu mô tả giá trị trong một con số dựa trên vị trí của nó trong số đó. Ví dụ, trong hệ thập phân, số 4 trong số 478 đại diện cho 400 đơn vị; số 4 trong số 547 đại diện cho 40 đơn vị. Cơ sở này sẽ được hiểu rõ hơn nếu chúng ta phân tích con số này như sau:

$$478 = 400 + 70 = 8$$

$$547 = 500 + 40 = 7$$

Trong bất kỳ hệ nào, giá trị của một số trong một số mô tả có thể được tính toán bằng cách sử dụng các bước sau: (1) Đánh số mũ cho các số từ phải sang trái. Bắt đầu là số 0 cho số phải nhất, và tăng lên 1 khi bạn di chuyển qua trái. (2) Sử dụng mỗi số mũ dưới dạng số mũ của cơ số đó. (3) Nhân giá trị của số nà với số mũ của cơ số tương ứng trong hệ thập phân.

Để tính toán ra số thập phân tương ứng, hãy cộng toàn bộ các kết quả vừa tính được ở bước 2.

***Ví dụ 1.7** Giá trị của mỗi con số trong số $1228_{(10}$ là bao nhiêu?*

Cơ số này là 10 bởi vì đây là số thập phân. Để tính giá trị của mỗi con số hãy theo các bước chỉ dẫn sau:

(1) Đánh số phía trên cho các số. Bắt đầu từ 0 ở vị trí phải nhất và tăng lên một khi di chuyển qua trái. Số này sẽ trông giống như sau:

$1^3 2^2 2^1 8^0$

(2-3) Sử dụng các số trên là lũy thừa của cơ số rồi nhân giá trị của nó cho số mũ tương ứng trong hệ thập phân:

Giá trị của 1 trog 1228 là bằng $1*10^3 = 1*1000 = 1000$.

Giá trị của 2 trước (tính từ trái sang phải) trong 1228 là bằng $2*10^2 = 2*100 = 200$

Giá trị của số 2 thứ hai (tính từ trái sang phải) trong 1228 là bằng $2*10^1 = 2*10 = 20$

Giá trị của 8 trong 1228 là bằng $8*10^0 = 8*1 = 8$.

***Ví dụ 1.8** Giá trị của mỗi con số trong số $1253_{(8}$ là bằng bao nhiêu? Hãy tính giá trị thập phân tương ứng của số này?*

Cơ số là 8 do đó số này sẽ được viết theo hệ bát phân. Sử dụng phương pháp tương tự ở ví dụ trên, chúng ta có thể sử dụng các số trên và viết số thứ tự là $1^3 2^2 5^1 3^0$. Sử dụng những số này dưới dạng là số mũ của cơ số, chúng ta được:

Giá trị của 1 trong 1253 là bằng $1*8^3 = 1*512 = 512$

Giá trị của 2 trong 1253 là bằng $2*8^2 = 2*64 = 128$

Giá trị của 5 trong 1253 là bằng $5*8^1 = 5*8 = 40$

Giá trị của 3 trong 1253 là bằng $3*8^0 = 3*1 = 3$

Giá trị thập phân tương ứng của số được tính bằng cách cộng tất cả các tích lại.

$$1253_{(8} = (1*8^3) + (2*8^2) + (5*8^1) + (3*8^0)$$
$$= (1*512) + (2*64) + (5*8) + (3*1)$$
$$= 512 + 128 + 40 + 3$$
$$= 683$$

Lưu ý rằng bất khi khi nào chúng ta tìm được giá trị thập phân tương đương, thì chúng ta cũng xác định được các giá trị cho mỗi số. Chúng ta sẽ sử dụng phương pháp kết hợp này kể từ bây giờ.

Ví dụ 1.9 *Giá trị của mỗi con số trong số $1AF_{(16}$ là bằng bao nhiêu? Giá trị thập phân tương đương của số này là bằng bao nhiêu?*

Cơ số này là 16 vì vậy số sẽ được viết theo dạng hệ thập lục phân. Bằng cách sử dụng các số ở trên chúng ta có thể viết số thứ tự là $1^3A^25^1F^0$. Để tính giá trị thập phân tương ứng cho số thập lục phân này, hãy thay các ký tự A và F bằng các giá trị thập phân tương ứng là 10 và 15. Giá trị thập phân tương ứng của số được tính bằng cách cộng tất cả các lũy thừa lại với nhau.

$$1A5F_{(16} = (1*16^3) + (10*16^2) + (5*16^1) + (15*16^0)$$
$$= (1*4096) + (10*256) + (5*16) + (15*1)$$
$$= 4096 + 2560 + 80 + 15$$
$$= 6751$$

Giá trị của 1 trong 1A5F là $1*16^3 = 1*4096 = 4096$

Giá trị của A trong 1A5F là $10*16^2 = 2*256 = 2560$

Giá trị của 5 trong 1A5F là $5*16^1 = 5*8 = 80$

Giá trị của F trong 1A5F là $15*16^0 = 3*1 = 15$

CHỦ ĐIỂM 1.5

CONVERSION BETWEEN THE BINARY, OCTAL, AND HEXADECIMAL SYSTEMS

Chuyển đổi giữa các hệ nhị phân, bát phân và thập lục phân

Table 1-3 shows the representation of the first fifteen decimal numbers in the binary, octal, and hexadecimal systems. Notice that each unique binary sequence is equivalent to one and only one symbol in each of these systems. We can use this equivalence between the binary, octal, and hexadecimal systems as a shorthand notation to convert values between any of these bases.

Table 1-3 Binary, Octal, Hexadecimal, and Decimal Equivalents.

Binary	Octal	Hexadecimal	Decimal
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

1.5.1 Conversion from Hexadecimal to Binary and Vice Versa

Given a hexadecimal number, we can find its binary equivalent by replacing each hexadecimal symbol by its corresponding binary configuration (see Table 1-3).

EXAMPLE 1.10 What is the binary equivalent of the hexadecimal number AF3B1?

Replacing each hexadecimal symbol by its corresponding binary sequence (see Table 1-3) we obtain

A	F	3	B	1
1010	1111	0011	1011	0001

Notice that to facilitate the reading of the resulting binary number we have separated it into four-bit groups.

To convert binary numbers to their hexadecimal equivalents, we can follow a two-step procedure that is almost the reverse of the previous process.

Step 1. Form four-bit groups beginning from the rightmost bit of the binary number. If the last group (at the leftmost position) has less than four bits, add extra zeros to the left of the bits in this group to make it a four-bit group.

Step 2. Replace each four-bit group by its hexadecimal equivalent (see Table 1-3).

EXAMPLE 1.11 What is the hexadecimal equivalent of the binary number shown below?

0110011110101010100111

Step 1. Forming four-bit groups beginning from the rightmost bit we have

01 1001 1110 1010 1010 0111

Since the last group (the leftmost) has only two bits, it is necessary to add two extra zero bits to the left of these bits to make it a four-bit group. The number should look like this:

0001 1001 1110 1010 1010 0111

Step 2. Replacing each group by its hexadecimal equivalent (see Table 1-3) we obtain

$$19EAA7(_{16}$$

1.5.2 Converting Decimal to Other Bases

The conversion of a given decimal number to another integer basis r (r > 0) is carried out by initially dividing the number by r, and then successively dividing the quotients by r until a zero quotient is obtained. The

decimal equivalent is obtained by writing the remainders of the successive divisions in the opposite order to that in which they were obtained. Example 1.12 illustrates this method.

EXAMPLE 1.12 What is the binary equivalent of decimal 41?

Since we want to convert decimal 41 to binary, we need to divide by the binary basis, that is, r = 2.

The initial number (41) is divided by 2 to obtain a quotient of 20 and a remainder of 1. The previous quotient of 20 is then divided by 2; this gives us a quotient of 10 and a remainder of 0. This new quotient of 10 is again divided by 2 to obtain 5 as the quotient and 0 as the remainder. The quotient of 5 is now divided by 2 to obtain a quotient of 2 and a remainder of 1. The new quotient is again divided by 2 to obtain a new quotient of 1 and a remainder of 0. This last quotient of 1 is divided by 2 again to obtain a quotient of 0 and a remainder of 1. Since we obtained a zero quotient the process stops. To form the binary number we write the remainders in the opposite order to that in which they were obtained, beginning with the last remainder. This process is shown below.

Number	Quotient When Dividing by 2	Remainder
41	20	**1**
20	10	**0**
10	5	**0**
5	2	**1**
2	1	**0**
1	0	**1**

The binary equivalent of decimal 41 is 101001. That is, $41_{10} = 101001_{2}$.

We can verify this result by converting 101101 to its decimal equivalent according to the procedure of Section 1.4.

$$\begin{aligned} 1^5 0^4 1^3 0^2 0^1 1^0 &= (1*2^5) + (0*2^4) + (1*2^3) + (0*2^2) + (0*2^1) + (1*2^0) \\ &= 32 + 0 + 8 + 0 + 0 + 1 \\ &= 41 \end{aligned}$$

EXAMPLE 1.13 What is the octal representation of decimal 41?

In this case we want to convert a decimal value to octal. Therefore, we need to divide by the octal basis, that is, by 8. Using the abbreviated method of the previous example we have that

Number	Quotient When Dividing by 8	Remainder
41	5	1
5	0	5

Therefore, $41_{(10} = 51_{(8}$.

We can verify this result by noting that $51_{(8} = (5*8^1) + (1*8^0)$
$= (40) + (1)$
$= 41_{(10}$

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 1.5

Bảng 1.3 minh họa kết quả của mười lăm số thập phân theo dạng nhị phân, bát phân và thập lục phân. Lưu ý rằng mỗi chuỗi nhị phân duy nhất tương ứng với một và chỉ một giá trị trong hệ khác. Chúng ta có thể sử dụng bảng tương đương giữa các hệ nhị phân, bát phân và thập lục phân như là một bảng sổ tay để chuyển các giá trị giữa bất kỳ cơ số nào.

Bảng 1.3 *Các giá trị tương đương giữa hệ nhị phân, bát phân, thập lục phân và thập phân.*

Nhị phân	*Bát phân*	*Thập lục phân*	*Thập phân*
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	13	A	10
1011	14	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

1.5.1 Chuyển từ hệ thập lục phân sang nhị phân và ngược lại

Cho một số thập lục phân, chúng ta có thể tìm được giá trị nhị phân tương ứng bằng cách thay thế mỗi ký tự thập lục phân bằng số nhị phân tương ứng của nó (xem bảng 1.3).

VÍ DỤ 1.10 *Số nhị phân tương ứng của số thập lục phân AF3B1 là bao nhiêu?*

Thay mỗi ký tự thập lục phân bằng một chuỗi nhị phân tương ứng (xem bảng 1.3), chúng ta được

A	F	3	B	1
1010	1111	0011	1011	0001

Lưu ý rằng để dễ đọc kết quả của số nhị phân chúng ta nên chia nó theo dạng các nhóm 4 bit.

Để chuyển các số nhị phân sang số thập lục phân tương ứng, chúng ta có thể thực hiện theo hai thủ tục như sau ngược lại với tiến trình trước.

Bước 1. *Từ nhóm 4 bit bắt đầu bên phải bit dài nhất của số nhị phân. Nếu nhóm cuối (nằm ở vị trí trái nhất) có ít hơn bốn bit, hãy thêm các bit zero vào ở bên trái của các bit trong nhóm này để tạo thành nhóm 4 bit.*

Bước 2. *Thay mỗi nhóm bốn bit này bằng giá trị thập lục phân tương ứng (xem bảng 1.3).*

VÍ DỤ 1.11 *Giá trị thập lục phân tương ứng của số nhị phân được trình bày dưới đây là bao nhiêu?*

01100111101010101100111

Bước 1. *Gom nhóm bốn-bit bắt đầu từ bit phải nhất chúng ta có*

01 1001 1110 1010 1010 0111

Bởi vì nhóm cuối (nhóm trái nhất) chỉ có hai bit, nó cần phải thêm hai bit zero vào bên trái để tạo thành nhóm bốn bit. Số này sẽ trông giống như dưới đây:

0001 1001 1110 1010 1010 0111

Bước 2. *Thay thế mỗi nhóm này bằng giá trị thập lục phân tương ứng (xem bảng 1.3) ta tính được*

$$19EAA7_{(16}$$

1.5.2 Chuyển hệ thập phân sang các cơ số khác

Chuyển đổi một số thập phân đã cho sang cơ số r (r > 0) được tính bằng cách lấy số ban đầu chia cho r và sau đó lấy thương số chia tiếp

cho r cho đến khi nào số thương là zero. Số thập phân tương đương được tính bằng cách viết các số dư của phép chia theo chiều ngược lại. Ví dụ 1.12 minh hoạ phương pháp này.

***VÍ DỤ 1.12** Số nhị phân tương ứng của số thập phân 41 là bao nhiêu?*

Bởi vì chúng ta muốn chuyển số thập phân 41 sang nhị phân, thì ta cần phải chia cho cơ số nhị phân, nghĩa là r = 2.

Số ban đầu (41) được chia cho 2 sẽ tính ra thương bằng 20 và số dư là bằng 1. Lấy số thương của 20 chia cho 2; chúng ta sẽ được số thương là 10 và số dư là 0. Lấy số thương mới này là 10 chia lại cho 2 và được kết quả là 5 và số dư là 0. Bây giờ lấy số thương 5 này chia cho 2 để được số thương bằng 2 và số còn dư là 1. Lấy số thương mới này lại chia cho 2 ta được kết quả số thương bằng 1 và số dư bằng 0. Lấy số thương cuối cùng là 1 chia cho 2 ta được kết quả số thương bằng 0 và phần dư bằng 1. Bởi vì chúng ta tính được số thương bằng zero do đó quá trình thực thi này dừng lại. Để biểu diễn số nhị phân chúng ta viết các số dư theo chiều ngược lại mà chúng ta tính toán bắt đầu từ số dư cuối cùng. Tiến trình này được minh họa dưới đây.

Số	Số thương khi chia cho 2	Phần dư
41	20	1
20	10	0
10	5	0
5	2	1
2	1	0
1	0	1

Số nhị phân tương ứng của thập phân 41 là 101001. Nghĩa là $41_{(10} = 101001_{(2}$.

Chúng ta có thể kiểm chứng kết quả này bằng cách chuyển 101101 sang dạng thập phân tương ứng dựa theo cách tính toán ở phần 1.4.

$$\begin{aligned} 1^5 0^4 1^3 0^2 0^1 1^0 &= (1*2^5) + (0*2^4) + (1*2^3) + (0*2^2) + (0*2^1) + (1*2^0) \\ &= 32 + 0 + 8 + 0 + 0 + 1 \\ &= 41 \end{aligned}$$

***VÍ DỤ 1.13** Số bát phân của số thập phân 41 là bao nhiêu?*

Trong trường hợp này chúng ta muốn chuyển giá trị thập phân sang bát phân. Do đó, chúng ta cần chia theo cơ số bát phân, đó là 8. Sử

dụng phương pháp tắt của của ví dụ trước chúng ta được

Số	Thương khi chia cho 8	Phần dư
41	5	**1**
5	0	**5**

41 5 1

5 0 5

Do đó $41_{(10} = 51_{(8}$

Chúng ta có thể kiểm chứng lại kết quả này bằng cách

$51_{(8} = (5*8^1) + (1*8^0)$

$= (40) + (1)$

$= 41_{(10}$

CHỦ ĐIỂM 1.6

RULES FOR FORMING NUMBERS IN ANY SYSTEM

Các quy tắc biểu diễn các số trong bất cứ hệ nào

Table 1-3 shows the equivalent of some numbers in binary, octal, and hexadecimal. In any of these systems, how do we form consecutive numbers higher than that represented by their largest individual symbol? In the decimal system, with a single digit, the largest number that we can form is nine. To represent numbers exceeding nine we use more than one digit as indicated below:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 . . . 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 . . .

After writing all single digits, we form all two-digit combinations beginning with 1. Then we form all two-digit combinations beginning with 2 and so on until we reach 99. After exhausting all two-digit combinations we start forming three-digit combinations, then we continue with all four-digits combinations and so on. This process can be continued forever. A similar procedure can be applied to other numerical systems. To form numbers higher than 7 in the octal system we use a similar formation rule as shown in Table 1-3.

EXAMPLE 1.14 In the hexadecimal system, how are the numbers greater than F formed?

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F . . . F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 100 101 . . . and so on.

Notice that, after writing all the single digits (shown in italics), it is necessary to form all possible two-digit combinations of the single digits, beginning with 1. Once this sequence has been exhausted, the two-digit combinations beginning with 2 are formed. This process is repeated until all two-digit combinations beginning with F are completed. At this moment, it is necessary to form all three-digit combinations, beginning with 100 and ending with FFF. The process is then repeated for all four-digit combinations and so on.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 1.6

Bảng 1.3 trình bày giá trị tương đương của các số trong hệ nhị phân, bát phân và thập lục phân. Trong bất kỳ hệ này, cách nào chúng ta hiển thị trình bày các số liên tục theo thứ tự số này cao hơn số trước bằng cách nào? Trong hệ thập phân với số đơn, số lớn nhất mà chúng ta biểu diễn là chín. Để mô tả các số lớn hơn chín chúng ta sử dụng nhiều số như được minh họa dưới đây:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 . . . 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 . . .

Sau khi viết tất cả các số đơn, chúng ta trình bày thành hai số kết hợp bắt đầu bằng 1. Sau đó chúng ta bắt đầu kết hợp hai số bằng 2 và ... cho đến 99. Sau khi đến cuối hai chữ số chúng ta bắt đầu biểu diễn bằng ba số kết hợp. Sau đó chúng ta tiếp tục bằng đầu bằng bốn số kết hợp và ..... Tiến trình này có thể được tiếp tục không dừng. Một tiến trình tương tự cũng có thể áp dụng cho các hệ số khác. Để trình bày các số lớn hơn 7 trong hệ bát phân chúng ta sử dụng quy tắc tương tự như được rình bày ở bảng 1.3.

VÍ DỤ 1.14 *Trong hệ thập lục phân, cách nào để biểu diễn các số lớn hơn F?*

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F . . . F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 100 101 . . .

Lưu ý rằng, sau khi viết tất cả các số đơn ra (xem các ký tự in nghiêng), nếu cần chúng ta có thể biểu diễn bằng hai số kết hợp, bắt đầu bằng 1. Khi chuỗi này kết thúc, hai số kết hợp sẽ bắt đầu bằng 2. Tiến trình này sẽ lặp lại cho đến khi nào hai số kết hợp bắt dầu là F. Ngay lúc này, chúng ta sẽ biểu diễn thành ba số kết hợp, bắt đầu là 100 và kết thúc là FFF. Tiến trình này lặp lại cho bốn số kết hợp và v,v.

CHỦ ĐIỂM 1.7

ARITHMETIC OPERATIONS IN THE BINARY, OCTAL, AND HEXADECIMAL SYSTEMS

Các phép toán số học trong các hệ nhị phân, bát phân và thập lục phân

Arithmetic operations on the binary, octal, and hexadecimal systems follow similar rules to that of the decimal system. To facilitate the explanation of arithmetic operations in any of these systems, let us review how these operations are carried out in the decimal system. Remember that in a + b, a is called the ***augend*** and b is the ***addend.*** Likewise in c - d, c is called the ***minuend*** and b is the ***subtrahend.***

EXAMPLE 1.15 In the decimal system the result of adding 452 and 385 is 837. How do we obtain this result?

```
452 +
385
---
  7 <-  Two plus five is seven. There is a single symbol for the number seven.

452 +
385
---
 37 <-  Five plus eight is thirteen. There is no single symbol for the number thirteen.
        Since 13 = 1*10 + 3 we write 3 and carry 1.
```

Notice that since thirteen is greater than nine we need more than one digit to represent thirteen. The rule for forming numbers (see Section 1.6) shows that, for number thirteen, 3 "accompanies" the number "one" that is carried.

```
1   <- carry
452 +
385
---
837 <-  One plus four is five. This five plus three is eight. There is a single symbol for
        eight.
```

EXAMPLE 1.16 When we subtract decimal 23 from decimal 4005 we obtain 3033. How do we obtain this result?

```
 4015
-982
------
    3 ← Five minus two is three. Therefore, write the symbol for three.

  11
 4015
-982
------
   33 ← Since one is less than eight we "borrow" one unit from the digit to the left of the
        one. The borrowed unit is equivalent to "borrowing" 10. This 10 plus 1 is equal to
        11. Therefore, write 3 since 11 - 8 = 3.

  9
 4015
-982
------
  033 ← The zero to the left of the one became a 9 (the basis of the decimal system minus
        1). Write 0 since 9 - 9 = 0.

  3
 4015
-982
------
 3033 ← The 4 to the left of the 0 "paid" the 10 that was previously "borrowed" by the 1. The
        4 decreases to 3.
```


1.7.1 Addition of Binary Numbers

The rules for adding or subtracting binary numbers are very similar to the ones used in the system. The major difference is that we are limited to only two digits. Table 1-4 shows the binary addition. To carry out these arithmetic operations properly it is necessary to align the n in their rightmost digit as we do in decimal arithmetic.

Table 1-4 Addition of Binary Numbers.

0+0=0
0+1=1
1+0=1
1 + 1 = 0 with a carry of 1

EXAMPLE 1.17 What is the result of adding binary numbers 10111100 and 11001111?

10111100 +
11001111

1 ← 0 plus 1 is 1. There is a symbol for 1.

10111100 +
11001111

11 ← 0 plus 1 is 1. There is a symbol for 1.

1 ← carry

10111100 +
11001111

011 ← 1 plus 1 is 0 with a carry of 1.

1
10111100 +
11001111

1011 ← 1 (the carry) plus 1 is 0 with a carry of 1. This 0 plus 1 is 1. Therefore, write 1 and carry 1.

1
10111100 +
11001111

01011 ← 1 (the carry) plus 1 is 0 with a carry of 1. This 0 plus 0 is 0. Therefore, write 0 and carry 1.

1
10111100 +
11001111

001011 ← 1 (the carry) plus 1 is 0 with a carry of 1. This 0 plus 0 is 0. Therefore, write 0 and carry 1.

1
10111100 +
11001111

0001011 ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 0 with a carry of one. Therefore, write 0 and carry 1.

```
 1
 10111100 +
 11001111
 ----------
110001011
```

— 1 (the carry) plus 1 is 0 with a carry of 1. This 0 plus 1 is 1. Therefore, write 1 and carry 1. Since there are no more digits to add we write the carry (in bold) as the leftmost digit.

The process of subtracting binary numbers is very similar to that of the decimal system. The main difference is that instead of "borrowing ten" we "borrow two" from the high-order digits (those to the left of a particular digit).

EXAMPLE 1.18 Subtract 11 from 1001 in binary.

```
1001
 -11
 ----
```

0 — 1 minus 1 is 0.

```
 1001
 -11
 ----
```

10 — 0 is less than 1. It is necessary to borrow "two" units from the higher unit. This two plus zero is two. Two minus one is one. Therefore, write 1.

```
  1
 1001
 -11
 ----
```

110 — The zero to the left became a 1 (the basis minus 1). This 1 minus the zero of the subtrahend (not shown here but assumed) is 1. Therefore, write 1.

```
  0
 1001
 -11
 ----
```

0110 — The leftmost 1 paid the "two" previously borrowed by the zero. This 1 becomes a zero. This zero minus the zero of the subtrahend (not shown here but assumed) is 0. Since this zero is the leftmost digit it is not generally written; however, we write it here for the sake of explanation.

1.7.2 Addition and Subtraction of Hexadecimal Numbers

Carrying out arithmetic operations in the binary system is an error-prone task since it is easy to get confused with so many ones and zeros. For this reason it is preferable to carry out arithmetic operations in the hexadeci-

mal system and then transform the results back to binary if necessary. The rules for adding and subtracting numbers in the hexadecimal system are similar to the ones used for the decimal system. However, we need to keep in mind that when performing additions there will be a carry whenever we exceed the value F. Likewise, when performing subtractions, if we need to "borrow" we always "borrow sixteen" instead of "borrowing ten" as we do in the decimal system. To simplify the process of doing arithmetic operations in the hexadecimal system it is convenient to think "in decimal" and then translate the results back to hexadecimal. The following example illustrates this process.

EXAMPLE 1.19 What is the result of adding the hexadecimal numbers 1A23 and 7C28?

```
1A23 +
7C28
------
   B  <-- Three plus eight is eleven. There is a symbol for eleven. Therefore, write B.

1A23 +
7C28
------
  4B  <-- Two plus two is four. There is a symbol for four. Therefore, write 4.

1A23 +
7C28
------
 64B  <-- Thinking "in decimal" we have that ten plus twelve is twenty-two. Since 22 = 16*1
          + 6, write 6 and carry 1.

 1
1A23 +
7C28
------
964B  <-- One (the carry) plus one is two. This two plus seven is nine. There is a symbol for
          nine. Therefore, write 9.
```

EXAMPLE 1.20 What is the result of subtracting in the hexadecimal system B2 from IFOOA?

```
1F00A
 -B2
------
   8  <-- Ten minus two is eight. There is a symbol for eight. Therefore, write 8
```

1F00A
−B2

58 ← Zero is less than eleven. Therefore, borrow "16" from the higher unit. This sixteen plus zero is sixteen. This sixteen minus eleven is five. Thus, write 5.

F
1F00A
−B2

F58 ← This zero became F (the basis minus 1). This F minus the zero of the subtrahend (not shown but understood) is F. Therefore, write E

E
1F00A
−B2

EF58 ← The F "paid" the 16 that was "borrowed" by the leftmost zero. The F became an E. This E minus the zero of the subtrahend (not shown but understood) is E. Therefore, write E.

1F00A
−B2

1EF58 ← One minus the zero of the subtrahend (not shown but understood) is 1. Therefore, write 1.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 1.7

1.7 CÁC PHÉP TOÁN SỐ HỌC TRÊN HỆ NHỊ PHÂN, BÁT PHÂN VÀ THẬP LỤC PHÂN

Các phép toán số học trên hệ nhị phân, bát phân và thập lục phân giống như quy tắc tính trong hệ thập phân. Để tiện lợi trong việc giải thích các phép tính số học trong bất kỳ các hệ, chúng ta hãy ôn lại cách mà các phép tính này trong hệ thập phân. Hãy nhớ rằng trong a + b, a được gọi là augend và b là addent. Tương tự như vậy trong c - d, c được gọi là minuend và b được gọi là subtrhend.

VÍ DỤ 1.15 *Trong hệ thập phân kết quả của phép cộng 452 và 385 là 837. Chúng ta tính kết quả này bằng cách nào?*

452 +
385

7 ← *Hai cộng năm bằng bảy. Số bảy là một số đơn*

```
  1
10111100 +
11001111
--------
  001011 ← (nhớ) cộng 1 bằng 0 nhớ 1. 0 cộng 0 bằng 0. Do đó, viết 0 nhớ 1.

 1
10111100 +
11001111
--------
 0001011 ← (nhớ) cộng 0 là 1. Số 1 cộng 1 bằng 0 nhớ 1. Do đó viết 0 nhớ 1

1
10111100 +
11001111
--------
110001011 ← 1 (nhớ) cộng 1 bằng 0 nhớ 1. 0 cộng 1 là 1. Do đó, viết 1 nhớ 1. Bởi vì không
            còn số khác để cộng do đó ta viết số trái nhất.
```

Quá trình trừ các số nhị phân giống như số thập phân. Khác biệt chính là thay vì mượn 10, chúng ta sẽ mượn 2 từ số ở vị trí cao hơn (nằm bên trái của số đặc biệt này).

***VÍ DỤ 1.18** Trừ 1001 cho 11 theo hệ nhị phân.*

```
1001
-11
----
   0 ← 1 trừ 1 bằng 0

  1001
  -11
  ----
    10 ← 0 nhỏ hơn 1. Do đó mượn 2 từ đơn vị cao hơn. Hai cộng zero là hai. Hai trừ
         một là một. Do đó, viết 1.

  1
  1001
  -11
  ----
   110 ← zero bên trái trở thành 1 (lấy cơ số này trừ 1). Số 1 này trừ cho zero của số
         bị trừ (được minh họa theo đề bài) là 1. Do đó, viết là 1.
```

1
1001
−11

0110 ← *Số trái nhất là 1 trả 'hai' đã mượn trước đó. Do đó 1 trở thành zero. Số zero này trừ zero cho số trừ (không được minh hoạ hưng giả định) là 0. Vì số 0 ở vị trí trái nhất do đó thường không được viết; tuy nhiên, chúng ta viết ở đây cho phù hợp với giải thích.*

1.7.2 Cộng và trừ các số thập lục phân

Việc tính toán các phép tính trong hệ nhị phân rất dễ sai bởi vì nó dễ lẫn lộn với nhiều số 1 và số 0. Vì lý do này chúng ta nên tính toán các phép số học trong hệ thập lục phân rồi chuyển kết quả sang hệ nhị phân nếu cần. Quy tắc cộng trừ các số hệ thập lục phân cũng giống như trong hệ thập phân. Tuy nhiên, chúng ta hãy nhớ rằng khi thực hiện phép cộng sẽ nhớ khi tính toán vượt quá giá trị F. Tương tự như vậy khi thực hiện phép trừ, nếu chúng ta cần mượn thì chúng ta luôn luôn mượn 16 thay vì mượn 10 như thực hiện trong hệ thập phân. Để đơn giản hóa quá tính toán trong hệ thập lục phân, chúng ta đang dùng hệ thập phân rồi chuyển kết quả sang hệ thập lục phân. Ví dụ sau đây minh họa cho quá trình này.

VÍ DỤ 1.19 *Kết quả của cộng hai số thập phân 1A 23 và 7C28 bằng bao nhiêu?*

1A23 +
7C28

B ← *Ba cộng tám bằng 11. Ký tự này là 11. Do đó, viết1 B.*

1A23 +
7C28

4B ← *Hai cộng hai là bốn . Ký tự này là 4, hãy viết 4.*

1A23 +
7C28

64B ← *Hãy nhớ rằng trong hệ thập phân, mười cộng mười hai là hai mươi hai. Bởi vì 22 = 16*1 + 6, viết 6 và nhớ 1.*

1
1A23 +
7C28

964B ← *Một (nhớ) cộng một là hai. Hai cộng bảy bằng chín. Ký tự này là chín. Do đó viết 9.*

VÍ DỤ 1.20 *Kết quả của trừ hai số thập phân 1F00A và B2 bằng bao nhiêu?*

```
 1F00A
  -B2
 ------
```

8 ← *Mười trừ hai là tám. Ký hiệu này là tám. Do đó, viết 8.*

```
 1F00A
  -B2
 ------
```

58 ← *Zero nhỏ hơn mười một. Do đó mượn 16 từ đơn vị cao hơn. Mười sáu cộng zero bằng 16. Mười sáu trừ mười một là năm. Do đó, viết là 5.*

```
   F
 1F00A
  -B2
 ------
```

F58 ← *Số zero này trở thành F (lấy số này trừ 1). Lấy F này trừ zero của số trừ (không hiển thị nhưng hểu ngầm) bằng F. Do đó, viết F.*

```
   E
 1F00A
  -B2
 ------
```

EF58 ← *F này trả 16 đã mượn. Do đó F trở thành E. Lấy E trừ zero của số trừ (không hiển thị nhưng ngầm hiểu) là E. Do đó viết E.*

```
 1F00A
  -B2
 ------
```

1EF58 ← *Một trừ zero của số trừ (không hiển thị nhưng ngầm hiểu) là 1. Do đó, viết 1.*

CHỦ ĐIỂM 1.8

REPRESENTING NUMBERS IN A COMPUTER
Trình bày các số trong một máy tính

All numerical data are represented inside the computer as a sequence of zeros and ones. The arithmetic operations, particularly the subtraction, raise the possibility that the result might be negative. How does the computer know whether a particular number represents a negative quantity or not? The answer to this question depends on the convention used to represent the numbers. In addition to the unsigned representation of a numerical quantity already discussed (see Section 1.1.1), there are some other conventions to represent both negative and positive numbers inside the computer. In this section we will discuss only two of these conventions (the sign-magnitude and the two's complement). An additional convention (one's complement) will be discussed later in the chapter (see solved problem 1.21). Any numerical convention needs to differentiate two basic elements of any given number: its sign (positive or negative) and the numerical value itself without the sign (the magnitude).

When representing numbers in any of these conventions the notion of "basic unit" needs to be stated explicitly since it may vary from manufacturer to manufacturer. In addition, it is necessary to choose one of the bits of the basic unit as the "sign bit." The leftmost bit is generally selected for this purpose. As any other bit, this bit can be 1 or 0. Computer manufacturers have agreed to use 0 as the "positive" sign and 1 as the "negative" sign.

1.8.1 The Sign-magnitude Convention

In this convention, given a basic unit of n bits, the leftmost bit is used exclusively to represent the sign. The remaining (n - 1) bits are used for the magnitude. Figure 1-2 shows that bit number 7 is considered the sign bit. The remaining digits (0 through 6) are used to represent the magnitude. The range of numbers that can be represented in this convention varies from $\pm(2^{n-1} + 2^{n-1} - 1)$

EXAMPLE 1.21 What is the sign-magnitude representation of the decimal numbers -41 and +41 if the basic unit is a byte?

Since the number +41 is positive we have to set the sign bit to 0. The remaining 7 bits will be used to represent the magnitude.

The binary representation of decimal 41 is given by the following binary sequence 0101001. That is, $41_{(10} = 0101001_{(2}$.

The sign-magnitude representation of the number is $00101001_{(2}$.

To represent the sign-magnitude of -41 we only need to change the sign bit from zero to 1. Since the binary representation of its magnitude is the same, we have that the sign-magnitude of -41 is $10101001_{(2}$.

EXAMPLE 1.22 What is the decimal equivalent value of the sign-magnitude binary sequence 10110111?

The number is negative since its sign (the leftmost bit) is 1.

The magnitude of the number is given by the sequence of bits 0110111. Using the procedure of Section 1.1.1 we find that

$$\begin{aligned} 0110111_{(2} &= (1*2^5) + (1*2^4) + (1*2^2) + (1*2^1) + (1*2^0) \\ &= 32 + 16 + 4 + 2 + 1 \\ &= 55 \end{aligned}$$

Therefore, the decimal equivalent of the number 10110111 represented in sign-magnitude is -55.

Addition of two numbers in sign-magnitude is carried out using the usual conventions of binary arithmetic. However, if both numbers have the same sign, we add their magnitude and copy the same sign. If the signs are different, we determine which number has the larger magnitude and subtract the other from it. The sign of the result is the sign of the operand with the larger magnitude.

As indicated before, given n bits, in sign-magnitude representation, the range of numbers that can be represented varies from $-2^{n+1}+ 2^{n-1}$. Therefore, if the result of an arithmetic operation falls outside that range, we will say that the operation causes an **overflow.**

EXAMPLE 1.23 What is the decimal value of the sum of the binary numbers 10100011 and 00010110 if they are represented in sign-magnitude? Assume that the basic unit is the byte.

Notice that the numbers have different signs: 10100011 is negative and 00010110 is positive. To calculate their sum, it is necessary to find out which of these two numbers has the larger magnitude. Using the procedure of Section 1.1.1 we have that:

Number: 10100011
Sign: 1 (negative)
Magnitude: 0100011 $= (1*2^5) + (1*2^1) + (1*2^0)$
$= 32 + 2 + 1$
$= 35$
Decimal value: -35

Number: 00010110
Sign: 0 (positive)
Magnitude: 0010110 = $(1*2^4) + (1*2^2) + (1*2^1)$
= 16 + 4 + 2
= 22
Decimal value: -35
Decimal value: +22

Value with the largest magnitude: 35 since 35 > 22.

Sign of the difference: negative (sign of the number with the larger numerical value)

Numerical value of the difference: 13(35 - 22 = 13)

Therefore, the decimal value of the sum is -13.

1.8.2 The Two's Complement Convention

The **two's complement convention or 2's complement** is the most popular among computer manufacturers since it does not present any of the problems of the sign-magnitude or one's complement (see solved problems *1.21* and *1.22).* Positive numbers are represented using a similar procedure to that of the sign-magnitude. Given n bits, the range of numbers that can be represented in two's complement is -2^{n-1} to $2^{n-1} - 1$. Notice that the range of negative numbers is one larger than the range of the positive values.

To represent a negative number in this convention, follow the three-step process shown below:

Step 1. Express the absolute value of the number in binary.

Step 2. Change all zeros to ones and all ones to zeros in the binary number obtained in the previous step. This process is called *"complementing the bits."*

Step 3. Add one to the binary number of step *2.*

In the two's complement convention, given a negative number, to find its positive counterpart we exercise steps *2* and 3 of the previous process.

EXAMPLE 1.24 What is the two's complement representation of -31?

Step 1. The absolute value of the number is 31. Using the procedure of Section 1.5.2, we have that the binary representation is 00011111(2. That is, $31_{(10} = 00011111_{(2}$.

Step 2. Changing all zeros to ones and vice versa, we have that the number looks like 11100000.

Step 3. Adding 1 we have that

```
11100000 +
       1
----------
11100001
```

Therefore, the two's complement representation of -31 is 11100001. Notice that the sign of the result is 1 as it should be.

EXAMPLE 1.25 What is the decimal positive value of the two's complement number 11100100?

Follow steps 2 and 3 of the procedure indicated above since the number is negative.

Step 2. Complement all bits of the given number. That is, change 11100100 to 00011011.

Step 3. Add 1 to the previous result and we have

```
00011011 +
       1
----------
00011100
```


Therefore, the positive counterpart of 11100100 is 00011100. The value of the latter number can be calculated using the procedure of Section 1.1.1. Thus, the decimal positive counterpart of the given number is 28.

1.8.2.1 Arithmetic Operations in Two's Complement

To add numbers represented in two's complement treat the numbers as unsigned integers. That is, treat the sign bit as any other bit of the number. Ignore any carry out of the leftmost position if there is one. To subtract two's complement numbers, treat the numbers as unsigned integers. If a borrow is needed in the leftmost place, borrow as if there were another bit to the left of the minuend.

EXAMPLE 1.26 What is the result of the addition of the numbers 11000111 and 11011101 if both numbers are represented in two's complement notation? What is the decimal value of the result?

```
11000111 +
11011101
----------
```

0 ← 1 + 1 = 0 with a carry of 1. Write 0 and carry 1.

```
      1
11000111 +
11011101
--------
      00 ← The carry plus one is zero with a carry of 1. This zero plus zero is zero. Write 0
           and carry 1.

      1
11000111 +
11011101
--------
     100 ← The carry plus one is zero with a carry of 1. The zero plus one is one. Therefore
           write one and carry 1.

    1
11000111 +
11011101
--------
    0100 ← The carry plus zero is one. This one plus one is zero with a carry of one. There-
           fore, write zero and carry one.

   1
11000111 +
11011101
--------
   00100 ← The carry plus zero is one. This one plus one is zero with a carry of one. Write
           0 and carry 1.

  1
11000111 +
11011101
--------
  100100 ← The carry plus zero is one. This one plus zero is one. Therefore, write 1.

11000111 +
11011101
--------
 0100100 ← One plus one is zero with a carry of one. Therefore, write zero and carry one.
 1

11000111 +
11011101
--------
10100100 ← The carry plus one is zero with a carry of one. This zero plus one is one.
           Therefore, write 1 and ignore the carry.
```

The decimal value of the result is -92. Note that -92 = -57 -35. Observe that 10100100 = -92, 11000111 = -57 and 11000111 = -35.

EXAMPLE 1.27 What is the result of subtracting 11011101 from 00111001? Assume that both numbers are represented in two's complement notation.

```
 00111001
-11011101
----------
```

0 – One minus one is zero. Therefore, write zero.

```
 00111001
-11011101
----------
```

00 – Zero minus zero is zero. Therefore, write zero.

```
 00111001
-11011101
----------
```

00 – Zero is less than one. Borrow "two" from higher unit. Since two minus one is one, write one.


```
   0
 00111001
-11011101
----------
```

1100 –The one (shown in bold and strikethrough) "paid" the unit that was borrowed by the zero. The one became a zero. Since this zero is less than one, borrow "two" from the higher unit. Therefore, write one since two minus one is one.

```
   0
 00111001
-11011101
----------
```

11100 –The one (shown in bold and strikethrough) "paid" the unit borrowed in the previous step. The one became a zero. Since zero is less than one, follow a process similar to the one described in the previous step. Therefore, write one.

```
   0
 00111001
-11011101
----------
```

011100 –The one (shown in bold and strikethrough) "paid" the unit borrowed in the previous step. The one became a zero. Since zero minus is zero. Therefore, write zero.

```
 00111001
-11011101
---------

 1011100 ← Zero is less than one. Borrow a "two" and write one since two minus one is one.

 1
 00111001
-11011101
---------

 01011100 ←The zero became a one. Write zero since one minus one is zero.
```

Notice that the previous operation assumes that there is an "invisible" one-bit to the left of the minuend. This "invisible" bit paid the 2's borrowed by the leftmost zeros.

1.8.2.2 Overflow Conditions in Two's Complement

As indicated before, given n bits, the range of numbers that can be represented in two's complement is -2^{n-1} to $2^{n-1}-1$. Whenever the result of an operation falls outside that range an overflow condition occurs. We will notice that in all overflow conditions, the sign of the result of the operation (addition or subtraction) is different than that of the operands. That is, if the operands are both positive, the result is negative or if the operands are both negative the result is positive. Solved problems 1.15 and 1.16 illustrate another method to detect if an overflow has occurred.

EXAMPLE 1.28 What is the result of adding the following two's complement numbers 11000111 and 10100100? 11000111 + 10100100

```
11000111 +
10100100
---------

      1 ← Write one since one plus zero is one.

11000111 +
10100100
---------

     11 ← Write one since one plus zero is one.

11000111 +
10100100
---------

    011 ← Write zero since one plus one is zero with a carry of one.
```

Table 1-5 Decimal Representation of Digits 0-9 in Two Different Weighted Codes

Decimal Digit	Weight 8-2-4-1	Weight 2-4-1-2
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
8	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 1 1

Two questions need to be answered concerning the dual representation of a digit. First, how does the computer know which one is the correct code? Second, can both be used interchangeably? The answer to the latter question is no. Only one code word can be used. However, this does not answer which of the two code words is the correct one. To answer this question satisfactorily we need to define the notion of a self-complementing code. A code is said to be self-complementing if, given any decimal digit N and its corresponding code word $X_1X_2X_3X_4$, the value of 9 - N (the 9's complement of the decimal) can be obtained by complementing the bits of the code word. We will use the notation X to indicate the complement of a bit X.

EXAMPLE 1.31 The code 2-4-2-1 is self-complementing. Notice that in this code the 9's complement of any decimal can be obtained by complementing the bits of its corresponding code word.

Decimal (N)	2-4-2-1	9 – N	Representation of 9 – N in 2-4-2-1
0	0 0 0 0	9	1 1 1 1
1	0 0 0 1	8	1 1 1 0
2	0 0 1 0	7	1 1 0 1
3	0 0 1 1	6	1 1 0 0
4	0 1 0 0	5	1 0 1 1
5	1 0 1 1	4	0 1 0 0
6	1 1 0 0	3	0 0 1 1
7	1 1 0 1	2	0 0 1 0
8	1 1 1 0	1	0 0 0 1
9	1 1 1 1	9	0 0 0 0

1.8.3.2 American Standard Code for Information Interchange

Computers are also used extensively to process alphanumeric information. To carry out this task the most common approach is to encode the individual characters that need to be manipulated. On a typical English computer keyboard there are at least *128* different characters:

Digits:	10
Letter (upper and lower):	52
Special symbol:	33 includes !, @, #, $, %, ^, &,*, (,) etc.
Control characters:	33 includes Enter, space, backspace, etc.
	128

The minimum number of bits necessary to represent these many characters is 7 since $2^7 = 128$. The most common code in the computer industry is the 7-bit code known as the American Standard Code for Information Interchange or ASCII for short. However, since the basic unit of storage is the byte, all ASCII codes are represented using eight bits. This leaves the most significant bit as zero. Another code used extensively by the International Business Machines Corporation is the Extended Binary Coded Decimal Interchange Code or EBCDIC for short. In this book we only consider the ASCII code. The set of ASCII characters is shown in Chapter *3*. **The ASCII code was** developed by the American National Standards Institute (ANSI) to allow information exchanges between equipment created by different manufacturers.

EXAMPLE 1.32 What is the ASCII representation of the message "Hello World"? Assume that addresses increase from left to right.

A sequence of characters is represented in memory as a series of consecutive bytes. Each byte holds one ASCII character code. Using the chart in Chapter 3 (see page 64) and the hexadecimal equivalent of each character the message representation is

H	e	l	l	o		W	o	r	l	d
48	65	6C	6C	6F	20	57	6F	72	6C	64

Observe that the space character (hex 20 or decimal 32) is a character like any other. However, the authors recognize that it is an elusive character that is very difficult to see. We have also assumed that the addresses increase from left to right to facilitate the reading of the characters.

1.8.4 Error Detection

When binary data are transmitted over any type of communication line, the possibility of an error in the transmission always exists due to equipment failure or the presence of "noise." When an error occurs it is possible that one or more bits of a byte could change from 0 to 1 or vice versa. Whenever this happens a code word can be changed into an incorrect but valid code or to a sequence of bits which do not represent anything. In this book we will only consider the detection and correction of single errors. That is, errors where only one single bit changes.

The numbers of bits that have to change within a byte before the byte is converted into an invalid code is sometimes used as a criterion for classifying codes. If only one bit of a byte needs to change, the code is said to be a *single-error-detecting code.* If only two bits need to change then the code is said to be a *two-error-detecting code.* To detect single errors we use an extra *parity check* bit. A parity bit is used to ensure that each code word, including the parity bit, has an even numbers of 1's (even parity) or an odd number of 1's (odd parity). Table 1-6 shows the even parity bit for the error-detecting code *8-2-4-1.*

Table 1-6

Decimal Digit	Weight 8-2-4-1	Parity Bit (even parity)
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	0
6	0 1 1 0	0
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	0

One of the most common tasks encountered in any computer system is that of transmitting information from one computer to another. The computer that sends the information is called the source; the computer that receives the information is called the destination. Both sender and receiver follow international standards to accomplish this task. It is assumed that the transmitted signal is divided into a series of one-bit intervals of length T During each of these intervals the source will send either a 0 or a 1 (shown in Fig. 1-8 as high and lows).

To communicate among themselves, computers may follow a convention like the one illustrated in Fig. 1-9. It is the responsibility of the destination to detect the value and correctness of the received signal.

The process that two computers may use to communicate with each other can be described as indicated below. We will use Fig. 1-9 as a reference.

(1) If the source is not sending any message, it continuously transmits a sequence of 1's.

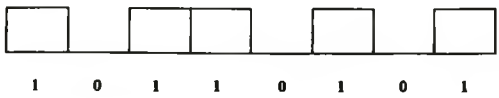


Fig. 1-8 Representation of 1 and 0 as high and lows.

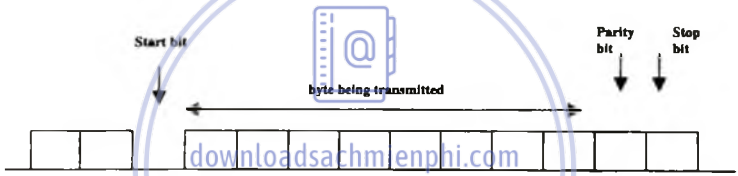


Fig. 1-9 Typical byte format for data transmission.

(2) A 0-bit indicates the beginning of the message. This first 0-bit is called the **start bit.** It is assumed that the bytes making up the message follow the start bit.

(3) The parity check bit is set according to the parity convention being used (even or odd).

(4) After the parity check bit another 1-bit (the **stop bit**) is transmitted to indicate that the transmission of the entire byte has been completed.

(5) At this point the source may continue transmitting the next byte or it may start transmitting a continuous sequence of 1's indicating that there are no more messages for the time being.

EXAMPLE 1.33 Two computers are communicating with each other using the data format convention of Fig. 1-9, the ASCII character set and even parity: The message shown below has a parity error. How does the computer recognize that an error has occurred? Can the computer tell the position of the complemented bit? Assume that the start bit is not shown. The arrow indicates the direction in which the characters need to be considered.

0101011001 0110100101 0110111101 0110110001 0110010101 0110010001
0111001111 0010000011 0110000111 0111001001 0110010101 0010000011
0110001011 0110110001 0111010111 0110010101 0010111001

Using the format of Fig. 1-9 each sequence of ten bits (one 8-bit byte, one parity check bit and one stop bit) can be considered as the representation of a single character of the message. Working with each group of bits separately and dropping the stop bit we can check the rightmost bit, the parity bit (shown in bold below).

1.	010101100 ✔	2.	011010010 ✔	3.	011011110 ✔	4.	011011000 ✔
5.	011001010 ✔	6.	011001000 ✘	7.	011100111 ✔	8.	001000001 ✔
9.	011000011 ✔	10.	011100100 ✔	11.	011001010 ✔	12.	001000001 ✔
13.	011000101 ✔	14.	011011000 ✔	15.	011101011 ✔	16.	011001010 ✔
17.	001011100 ✔						

The parity check bit of group No. 6 should be one instead of zero since we are working with even parity. Observe that the computer cannot tell the position of the erroneous complemented bit. Use the chart in Chapter 3 to decode the message. Can you guess the word that was changed? Can you tell the bit that was complemented? We leave these questions as an exercise for the reader.

The previous example shows that using the parity bit alone the computer can tell that an error has occurred but cannot tell the position of the bit that was complemented.

In general, given n bits, to obtain an error-detecting code no more than half of the 2^n possible combinations of these n bits can be used. In addition, the code words must be chosen in such a way that, for any code word to produce another valid word, at least two bits must be complemented. The minimum number of bits that need to change in a code word to produce another valid coded word is called the ***minimum distance*** of the code. Therefore, we can rephrase the previous statement by saying that given n bits, to obtain an error-detecting code its minimum distance must be two or more. The even parity 8-4-2-1 code shown in Table 1-6 has a minimum distance of two.

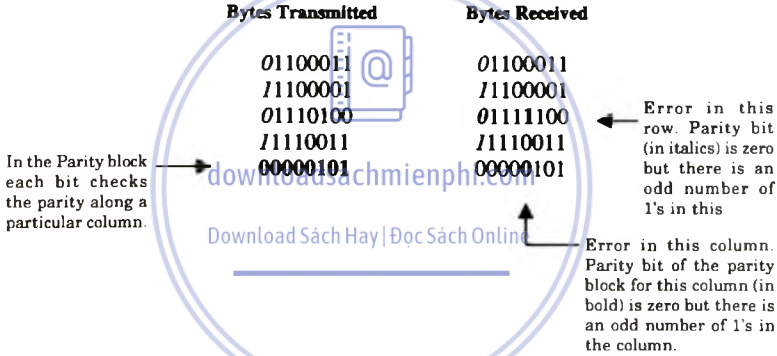
1.8.5 Error Correction

Once an error has been detected, there are some methods that can be used to identify and correct the complemented bit. In this section we will consider some of the error-correcting codes. In general, a code is said to be an ***error-correcting code*** if the correct code word can always be inferred from the erroneous code. The Forward Error Correction (FEC) and the Hamming Code methods are examples of this type of code. Both methods require additional redundant bits to identify and correct the errors.

1.8.5.1 Forward Error Correction

In this correction schema, a **parity block** complements the parity bit of each byte after a predetermined number of n bytes. Each bit of the parity block checks the parity of the preceding n bits that occupy the same position in the preceding n bytes. Using both pieces of information it is possible to find and correct the bit in error. Example 1.34 illustrates this method and the use of the msb of an ASCII code as a parity bit.

EXAMPLE 1.34 Assume that two computers communicate with each other using ASCII code and that a parity block is transmitted every four bytes. In addition, consider that even parity is used at both the byte level and the block level. Parity bits are shown in italics and the parity block is shown in bold.



It is the responsibility of the receiving computer to check the parity of every column and every row. Whenever the computer identifies the bit in error in a particular byte it complements the bit to recover from the error.

1.8.5.2 Hamming Code

This method can be used to provide multiple-error detection. The fundamental principles in constructing this code for m given bits are as follows:

(1) Add k additional *parity checking bits* denoted by p_1, p_2, ... pk to the m given bits. The resulting code will have code words of (m + k) bits. Choose the value of k so that it satisfies the inequality

$$2^k \geq m + k + 1$$

For example, if we want to transmit four data bits (m = 4), the value of k must be 3 since

$$2^3 \geq 4 + 3 + 1$$

This result implies that we need to add three parity checking bits denoted by p_1, p_2, and p_3.

(2) Number the (m + k) bits 1 through m + k; start by assigning 1 to the most significant bit and continue until you assign the value m + k to the least significant bit. Consider these numbers as the positions of the bits in a code word.

(3) Place the checking bits in positions 1, 2, 4 . . . 2^{k-1} of the code word. When k = 3, the checking bits are placed in positions 1, 2, and 2^{3-1}. That is, in positions 1, 2, and 4.

The position of the bits b_1, b_2, b_3, and b_4 and the checking bits p1, p2, and p3 in the encoded word is as follows:

Position →	1	2	3	4	5	6	7
	p_1	p_2	b_1	p_3	b_2	b_3	b_4

(4) Form k sets, P_1, P_2 . . . P_k of binary numbers. The binary numbers in set Pi should be such that their representation has k or fewer bits and has 1 in the jth, position. Select p; in such a way that it has even parity in the positions indicated by the elements of the set Pj.

For k = 3 form sets P_1, P_2, and P_3. The elements of each of these sets should have 3 or fewer bits in their representation. Table 1-7 shows the seven possible combinations that we can form with 3 bits that satisfy the condition of step 4. According to this table the sets and their elements are:

P1 = {1,3,5,7} P2 = {2,3,6,7} P3 = {4,5,6,7}

Notice that the elements of P1 have 1's in column 1, the elements of P_2 have 1's in column 2, and the elements of P_3 have 1's in column 3.

Table 1-7

	P_3	P_2	P_1
1.	0	0	1
2.	0	1	0
3.	0	1	1
4.	1	0	0
5.	1	0	1
6.	1	1	0
7.	1	1	1

Therefore,

select p_1 so as to establish even parity in positions 1, 3, 5, and 7.
select p_2 so as to establish even parity in positions 2, 3, 6, and 7.
select p_3 so as to establish even parity in positions 4, 5, 6, and 7.

(5) Form the code word to be transmitted by adding the appropriate checking digits so that the parity condition of the previous step is satisfied.

EXAMPLE 1.35 If the sequence 1101 (data) is to be transmitted, what is the code word if Hamming Code is used?

Position	1	2	3	4	5	6	7
	p_1	p_2	b_1	p_3	b_2	b_3	b_4
Transmitted data			1		1	0	1
Even parity in positions 1, 3, 5, 7 requires that $p_1 = 1$	1		1		1	0	1
Even parity in positions 2, 3, 6, 7 requires that $p_2 = 0$	1	0	1		1	0	1
Even parity in positions 4, 5, 6, 7 requires that $p_3 = 0$	1	0	1	0	1	0	1

The code word is formed by concatenating the value of the bits in positions 1 through 7. In this case the actual word being transmitted is 1010101.

Note: The reader should be aware that depending on how the bits are numbered we may obtain a different value. In this section, as indicated in step 2, we assign the value 1 to the most significant bit and the value 7 to the least significant bit. The code will work with either convention as long as the reader is consistent in using the numbering scheme.

To locate and correct the error use the following steps:

(1) Perform k parity checks on selected digits of each code word. The result of each of these parity checks is either 0 if no error has occurred or 1 if an error has been detected.

(2) Using the results of the parity tests, form a binary number $r_k r_2 \ldots r_1$. The decimal value of this number gives the position of the erroneous digit.

EXAMPLE 1.36 Assume that the word sent is 1010101. If the word received is 1011101, find the position of the bit in error using the procedure indicated above.

(1) Since three checking digits were added, three parity checks need to be performed.

Position	1	2	3	4	5	6	7	
	p_1	p_2	b_1	p_3	b_2	b_3	b_4	
Data	1	0	1	1	1	0	1	
parity check for p_1 =	1		1		1		1	r1 = 0 since parity is even
parity check for p_2 =		0	1			0	1	r2 = 0 since parity is even
parity check for p_3 =				1	1	0	1	r3 = 1 since parity is odd

The position of the erroneous bit is $r_3r_2r_1$ = 100. The decimal value of this number is $1^20^10^0 = 1*2^2 = 4$. This result indicates that the erroneous bit occupies position 4 on the received byte as we have already assumed. The bit on the 4th position is then complemented to form the correct message.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 1.8

1.8 TRÌNH BÀY CÁC CON SỐ TRONG MÁY TÍNH

Tất cả các dữ liệu số được trình bày bên trong máy tính dưới dạng một chuỗi các số zero và số 1. Các phép toán học đặc biệt là phép trừ có thể có kết quả âm. Làm sao máy tính hiểu được một số đặc biệt mô tả cho số âm? Câu trả lời này phụ thuộc vào nguyên tác sử dụng để trình bày các con số. Cách mô tả một số không dấu đã được thảo luận (xem phần 1.1) có nhiều quy tắc trình bày các số dương và âm trong máy tính. Trong phần này chúng ta sẽ thảo luận hai quy tắc (dấu độ lớn và phần bù của hai). Một quy tắc bổ sung (phần bù của một) sẽ được thảo luận sau chương này. (xem bài tập 1.21). Bất kỳ quy tắc số nào cũng có hai thành phần khác biệt cơ bản của một số đã cho: dấu của nó (dương hoặc âm) và giá trị không dấu (độ lớn).

Khi biểu diễn các số trong các quy tắc này, khái niệm về đơn vị cơ bản cần được phát biểu chính xác bởi vì nó có thể khác nhau tùy theo nhà sản xuất. Nó cần phải chọn một trong các bit của đơn vị cơ bản là bit tín hiệu. Bit trái nhất được chọn cho mục đích này. Các bit khác có thể 1 hoặc 0. Các nhà sản xuất máy tính đồng ý sử dụng 0 như là ký hiệu "dương" và 1 là ký hiệu âm "âm".

1.8.1 Quy ước về dấu độ lớn

Trong quy ước được cho bởi một đơn vị cơ bản là n bit, bit trái nhất được dùng để mô tả tín hiệu. Các bit còn lại (n - 1) được dùng để mô tả độ lơn. Hình 1.2 minh họa bit số 7 là bit tín hiệu. Các số còn lại (0 đến 6) là bit được dùng để mô tả độ lớn. Các số được mô tả trong quy tắc này nằm trong $2^{n+1} + 2^{n-1} - 1$.

VÍ DỤ 1.21 *Trình bày dấu độ lớn của các số thập phân - 41 và + 41 nếu đơn vị cơ bản là byte?*

Bởi vì số + 41 là số dương do đó ta sẽ đặt bit ký hiệu là 0. 7 bit còn lại sẽ được sử dụng để mô tả độ lớn.

Kiểu trình bày nhị phân của số thập phân 41 được cho bởi chuỗi thập phân 0101001. Nghĩa là $41_{(10} = 0101001_{(2}$

Trình bày dấu độ lớn của số này là $00101001_{(2}$

Để trình bày dấu độ lớn của 41 ta chỉ cần thay đổi bit tín hiệu từ zero sang 1. Vì mô tả độ lớn của số nhị phân là như nhau, do đó ta có được tín hiệu của độ lớn của - 41 là $10101001_{(2}$

VÍ DỤ 1.22 *Giá trị thập phân tương ứng của tín hiệu độ lớn của chuỗi nhị phân 1011011 là bao nhiêu?*

Số này là số âm bởi vì tín hiệu của nó (ở bit trái nhất) bằng 1.

Độ lớn của số được cho bởi chuỗi của bit 0110111. Bằng cách sử dụng phương pháp ở phần 1.1.1 ta tìm được.

$$\begin{aligned} 0110111_{(2} &= (1*2^5) + (1*2^4) + (1*2^2) + (1*2^1) + (1*2^0) \\ &= 32 + 16 + 4 + 2 + 1 \\ &= 55 \end{aligned}$$

Do đó, số thập phân tương ứng của số 10110111 được mô tả theo tín hiệu độ lớn là - 55.

Cộng hai số trong tín hiệu độ lớn được tính bằng cách sử dụng các phép tính nhị phân thông thường. Tuy nhiên, nếu hai số này cùng dấu, chúng ta cộng độ lớn rồi sao chép dấu đó, nếu các dấu khác nhau, chúng ta xác định số nào lớn hơn và trừ cho số còn lại. Dấu của kết quả là dấu của toán hạng có độ lớn hơn.

Như đã trình bày ở trước, cho n bit, trong mô tả tín hiệu độ lớn, các số được trình bày nằm trong đoạn $2^{n+1} + 2^{n-1} - 1$. *Do đó, nếu kết quả của phép toán nằm ngoài dãy này, ta nói phép toán này là nguyên nhân gây ra tràn bộ nhớ.*

VÍ DỤ 1.23 *Giá trị thập phân của tổng hai số nhị phân 10100011 và 00010110 là bao nhiêu nếu chúng được mô tả theo dấu độ lớn? Giả sử rằng đơn vị cơ bản là byte.*

Lưu ý rằng các số có các ký hiệu khác nhau là: 10100011 là số âm và 00010110 là dương. Để tính toán tổng, chúng ta cần tìm số lớn hơn. Bằng cách sử dụng phương pháp ở phần 1.1.1 ta có được.

Số: 10100011 *Số : 00010110*

Ký hiệu: 1 (âm) *Ký hiệu: 0 (dương)*

Độ lớn: $0100011 = (1*2^5) + (1*2^1) + (1*2^0)$
$= 32 + 2 + 1$
$= 35$

Độ lớn $0010110 = (1*2^4) + (1*2^2) + (1*2^1)$
$= 16 + 4 + 2$
$= 22$

Giá trị thập phân:- 35 *Giá trị thập phân: + 22*

Giá trị lớn nhất của: 35 vì 35 > 22

Ký hiệu khác nhau: âm (ký hiệu của số có giá trị lớn hơn)

Tính ra giá trị khác: 13(35 - 22) = 13)

Do đó, giá trị thập phân của tổng là - 13.

1.8.2 Quy tắc phần bù của hai

Quy tắc phần bù của hai hoặc phần bù 2 là phép tính phổ biến nhất của máy tính bởi vì nó không gặp bất kỳ vấn đề gì cũng như quy tắc dấu độ lớn hoặc phần bù của 1 (xem bài tập 1.21 và 1.22). Các số dương được mô tả giống như dấu độ lớn. Cho n bit, dãy các số được miêu tả trong phần bù của hai là 2^{n-1} *cho đến* $2^{n-1} - 1$*.Lưu ý rằng dãy số âm lớn hơn giá trị của dãy số dương.*

Để mô tả một dãy số âm trong quy tắc này, hãy theo ba bước sau:

Bước 1. Biểu diễn giá trị tuyệt đối của số này dưới dạng nhị phân.

Bước 2. Thay đổi tất cả các số zero bằng 1 và tất cả số 1 thành zero trong số nhị phân đã tính trong bước trước. Quá trình này được gọi là "phần bù của bit".

Bước 3. Thêm cộng 1 vào số nhị phân ở bước 2.

Trong quy tắc phần bù của hai, với một số âm đã cho, để tìm giá trị dương tương ứng, ta thực hành các bước 2 và 3.

***VÍ DỤ 1.24** Phần bù của hai bằng bao nhiêu ứng với số 31?*

Bước 1. Lấy trị tuyệt đối của số này bằng 31. Bằng cách sử dụng phương pháp ở phần 1.5.2, chúng ta có được số nhị phân tương ứng là $00011111_{(2}$*. Nghĩa là* $31(10_2 = 00011111_2$

Bước 2. Thay đổi tất cả số zero thành 1 và ngược lại, chúng ta có được số 11100000.

Bước 3. Cộng 1 chúng ta có được

```
11100000 +
       1
----------
11100001
```

Do đó, phần bù hai của số 31 là 11100001. Lưu ý rằng dấu của của kết quả là 1.

VÍ DỤ 1.25 *Giá trị dương thập phân của phần bù hai là 11100100 là bao nhiêu?*

Tuân theo các bước 2 và 3 của phương pháp chỉ dẫn ở trên thì số này là âm.

Bước 2. Lấy phần bù cho các bit của số đã cho. Do đó, nó sẽ đổi là 11100100 thành 00011011.

Bước 3. Cộng 1 vào kết quả ta có

```
00011011 +
       1
----------
00011100
```


Do đó, phần dương tương ứng của 11100100 là 00011100. Giá trị của số sau có thể được tính toán bằng cách sử dụng phương pháp ở phần 1.1.1. Do đó, kết quả số dương của số đã cho là 28.

1.8.2.1 Các phép tính số học trong phần bù của hai

Để cộng các số được mô tả trong phần bù của hai hãy xem các số này là các số nguyên không dấu. Điều này có nghĩa là hãy xem các ký hiệu bit như là một số nguyên khác. Bỏ qua bất kỳ giá trị nhớ nào ở vị trí trái nhất nếu có. Để trừ các số bù, hãy xem các số này là số nguyên không dấu. Nếu cần mượn thêm tại vị trí bên trái, hãy mượn nếu có một bit khác nằm bên trái.

VÍ DỤ 1.26 *Kết quả của phép cộng các số 11000111 và 11011101 bằng bao nhiêu nếu cả hai số này được mô tả dưới dạng phần bù của hai? Giá trị thập phân này bằng bao nhiêu?*

```
11000111 +
11011101
----------
```

0 — 1 + 1 = 0 *với 0 nhớ 1. Viết 0 nhớ 1*

```
 11000111 +
 11011101
 --------
       00 ←
```
Lấy số nhớ cộng với 1 bằng 0 nhớ 1. 0 cộng 0 bằng 0. Viết 0 nhớ 1.

```
    1
 11000111 +
 11011101
 --------
      100 ←
```
Lấy số nhớ cộng với 1 bằng 0 nhớ 1. Lấy 0 cộng 1 bằng 1. Do đó viết 1 nhớ 1.

```
   1
 11000111 +
 11011101
 --------
     0100 ←
```
Lấy số nhớ cộng với 0 bằng 1. Lấy số 1 cộng với 1 bằng 0 nhớ 1. Do đó, viết 0 nhớ 1.

```
  1
 11000111 +
 11011101
 --------
    00100 ←
```
Lấy số nhớ cộng 0 bằng 1. Lấy 1 cộng với 1 bằng 0 nhớ 1. Viết 0 nhớ 1.

```
 1
 11000111 +
 11011101
 --------
   100100 ←
```
Lấy số nhớ cộng với 0 bằng 1. Lấy 1 cộng 0 bằng 1. Do đó viết 1.

```
 11000111 +
 11011101
 --------
  0100100 ←
```
Một cộng một bằng 0 nhớ 1. Do đó, viết 0 nhớ 1.

```
1
 11000111 +
 11011101
 --------
 10100100 ←
```
Lấy nhớ cộng 1 là 0 nhớ 1. Lấy 0 này cộng với 1 là 1. Do đó viết 1 và bỏ phần nhớ.

Giá trị thập phân của kết quả này là 92. Lưu ý rằng 92 = 57 + 35. Do đó 101000100 = 92, 11000111 = 57 và 11000111 = 35.

VÍ DỤ 1.27 *Kết quả của phép trừ 11011101 từ 00111001 là bao nhiêu? Giả sử cả hai số này được trình bày dưới dạng phần bù của hai.*

```
  00111001
- 11011101
----------
         0 ←  1 trừ 1 bằng 0. Do đó viết 0.

  00111001
- 11011101
----------
        00 ←  0 trừ 0 bằng 0. Do đó viết 0.

  00111001
- 11011101
----------
        00 ←  0 nhỏ hơn 1, do đó mượn "hai" từ đơn vị cao hơn. Do đó 2 trừ 1 là 1, viết
              1.

    0
  00114001
- 11011101
----------
      1100 ←  1 (được in đậm và kẻ ngang) "trả lại" cho số đã mượn từ zero trước đó. 1
              trở thành 0. Bởi vậy 0 nhỏ hơn 1, mượn "hai" tính từ đơn vị cao hơn. Do
              đó, viết một từ 2 trừ 1 là 1.

    0
  00141001
- 11011101
----------
     11100 ←  1 (được in đậm và kẻ ngang) "trả lại" cho số đã mượn từ zero trước đó. 1
              trở thành 0. Vì 0 nhỏ hơn 1, thực hiện theo qui trình tương tự như qui
              trình đã được mô tả trong bước trước đó. Do đó, viết 1.

    0
  00411001
- 11011101
----------
    011100 ←  1 (được in đậm và kẻ ngang "phải trả" cho số được mượn ở bước trước. 1
              trở thành 0. Bởi vì 0 trừ 0. Do đó, viết 0.

  00111001
- 11011101
----------
   1011100 ←  0 nhỏ hơn 1. Mượn "hai" và viết 1 vì 2 trừ 1 là 1.

  1
  00111001
- 11011101
----------
  01011100 ← zero trở thành 1. Viết 0 bởi vì 1 - 1 là zero.
```

Lưu ý rằng giả sử phép toán ở trước có một bit không nhận thấy nằm ở bên trái của số bị trừ. Bit không nhìn thấy này sẽ trả hai được mượn bởi zero trước đó.

***1.8.2.2** Các điều kiện tràn bộ nhớ trong phần bù của hai.*

Như hướng dẫn ở phần trước với một số n đã cho, dãy các số được biểu diễn theo hần bù của hai là 2^{n-1} đến $2^{n-1} - 1$. Cuối cùng kết quả của mỗi phép toán nằm ngoài dãy này thì điều kiện tràn bộ nhớ xảy ra. Chúng ta thấy rằng trong tất cả điều kiện tràn bộ nhớ dấu kết quả của toán tử (cộng hoặc trừ) khác với các toán hạng khác. Điều này là vì nếu các toán hạng lại dương thì kết quả là âm hoặc nếu các toán hạng âm thì kết quả là dương. Các bài tập 1.15 và 1.16 sẽ chỉ ra phương pháp khác để phát hiện việc tràn bộ nhớ có xảy ra hay không.

***VÍ DỤ 1.28** Nếu kết quả của phép cộng số bù của hai là 11000111 và 10100100 là bằng bao nhiêu?*

```
11000111 +
10100100
--------

      1 ← Viết 1 bởi vì 1 cộng 0 là 1

11000111 +
10100100
--------

     11 ← Viết 1 bởi vì một 1 cộng 0 là 1

11000111 +
10100100
--------

    011 ← Viết 0 bởi 1 cộng 1 là zero nhớ 1.

   1
11000111 +
10100100
--------

   1011 ← Phần nhớ cộng 0 là 1. 1 cộng 0 là 1. Vì vậy viết 1.

11000111 +
10100100
--------

  01011 ← 0 cộng 0 là 0. Do đó,v viết 0.

11000111 +
10100100
--------

 101011 ← 0 cộng 1 là 1. Viết 1.
```

11000111 +
10100100

1101011 — *1 cộng 0 là 1. Do đó viết 1.*

11000111 +
10100100

01101011 — *1 cộng 1 là 0 nhớ 1. Do đó, viết 1 và bỏ qua phần nhớ.*

Lưu ý rằng một số tràn đã xảy ra bởi vì kết quả của phép cộng là số dương trong khi các toán hạng đều là số âm. Giá trị này cho thấy giá trị của nó nằm ngoài dãy cho phép trong một byte theo phần bù của hai. Trong thực tế, với 8 bit các giá trị biến đổi từ -2^7 đến $2^7 - 1$ hoặc từ - 128 đến 127 (xem phần 1.8.2). Giá trị tổng theo số thập phân tính là - 149. Giá trị này nằm ngoài dãy - 128 đến 127.

1.8.3 Các mã nhị phân và các mã chữ số

Con người thích làm việc với các số thập phân hơn với chuỗi dài các số 0 và 1. Máy tính làm việc với các chuỗi nhị phân. Để tạo cầu nối trong việc giao tiếp giữa con người và máy móc, nhiều mã số được tạo ra nhiều đoạn mã số được tạo ra để các số thập phân được mô tả theo một chuỗi các số nhị phân. Bằng cách này máy tính có thể thực thi tất cả tính toán của nó trong hệ nhị phân rồi sau đó chuyển kết quả này sang dạng thập phân cho ứng dụng của con người. Trong phần này chúng ta sẽ xem xét một số mã quan trọng (BCD và Hamming) và các mã số ASCII.

Một mã nhị phân là một nhóm n bit có 2^n kết hợp khác nhau của 0 và 1 với mỗi tổ hợp biểu thị cho một thành phần của một tập hợp đang được mã hóa. Ví dụ với 2 bit có thể biểu diễn một tập hợp gồm bốn phần tử khác nhau. Có nghĩa là chúng ta có thể biểu diễn bốn phần tử khác nhau trong một tập hợp. 2^2 tổ hợp khác nhau mà chúng ta có thể với diễn với hai bit là: 00, 01, 10, và 11. Chúng ta có thể liên kết mỗi tổ hợp này với mỗi phần tử riêng biệt của tập hợp khác. Khi đó chúng ta nói rằng mỗi phần tử "đại diện" cho một phần tử của tập hợp. Để mô tả một tập hợp có tám phần tử chúng ta chỉ cần ba bit ; để mô tả 16 phần tử chúng ta cần bốn bit. Nếu số phần tử cần mô tả không phải là lũy thừa của hai thì mã nhị phân này sẽ có các các tổ hợp bit không xác định. Vấn đề này được minh họa như sau.

VÍ DỤ 1.29 *Có bao nhiêu bit cần để mô tả số 10 trong chữ số thập phân?*

Bởi vì $2^3 < 10 < 2^4$ do đó giá trị nhỏ nhất của bit để mô tả số 1 = 4. Tuy nhiên, vì $2^4 = 16$, do đó có 6 (=16 - 10) tổ hợp có 4 bit , được gán vào một số thập phân riêng biệt.

1.8.3.1 Các mã tăng trọng

Để mô tả các chữ số thập phân bằng cách sử dụng bốn bit các mã phổ biến nhất là các mã tăng trọng. Trong loại mã này mỗi bit được gán cho một "trọng lượng". Giá trị của mỗi chữ số thập phân được trình bày theo tổ hợp của các bit được xác định bằng cách nhân mỗi bit này c ıo trọng lượng của nó và cộng các tích riêng biệt này lại. Nếu w_1, w_2, w_3 và w_4 tương ứng với trọng lượng của các bit b_1, b_2, b_3 và b_4 sau đo 1số thập phân N được trình bày là

$$N = w_1b_2 + w_2b_2 + w_3b_3 + w_4b_4$$

Một chuỗi số nhị phân mô tả cho một số thập phân được gọi là một mã từ. Bảng 1.5 trình bày hai mã nhị phân. mã này với trọng lượng 8, 4, 2 và 1 được biết như là mã BCD (Binary-Coded-Decimal). Mỗi mã từ trong BCD là một số nhị phân tương ứng với số mà nó hiển thị.

***Bảng 1.5** Bảng trình bày số thập phân từ 0 - 9 theo hai mã tăng trọng khác nhau*

Số thập phân	Trọng lượng 8-2-4-1	Trọng lượng 2-4-1-2
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
8	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 1 1

***VÍ DỤ 1.30** Trong mã 24-2-1, phần mô tả của số thập phân 9 được cho là 1111. Lưu ý rằng.*

$$1*2 + 1*4 + 1*2 + 1*1 = 9$$

Điều thú vị khi quan sát các mã này là cách trình bày của các số thập phân có thể không giống nhau. Hãy xem mã 2-4-2-1 chúng ta có thể trình bày số thập phân 7 theo hai cách khác nhau:

$$1*2 + 1*4 + 0*2 + 1*1 = 7 \text{ và } 0*2 + 1*4 + 1*2 + 1*1 = 7$$

Có hai câu hỏi quan tâm đến việc cách trình bày kép (dual) của một chữ số. Trước tiên, làm cách nào máy tính nhận ra mã nào là đúng? Câu hỏi thứ hai cả hai cách này có thể được dùng thay thế cho nhau

không? Câu trả lời cho câu hỏi thứ hai là không. Chỉ có một mã từ mới được dùng. Điều này không phải là câu trả lời cho hai mã từ chỉ một cái đúng. Để trả lời thỏa đáng câu hỏi này, chúng ta cần phải định nghĩa khái niệm của một mã tự bù (self-complementing code). Một mã được gọi là tự bù nếu cho bất kỳ số thập phân N nào và có mã từ tương ứng $X1_zX2_zX3_zX4_z$, thì giá trị của 9 - N (phần bù 9 của số thập phân này có thể xác định) bằng cách lấy phần bù theo các bit của mã từ. Chúng ta sẽ sử dụng khái niệm X để chỉ ra phần bù của X.

VÍ DỤ 1.31 *Mã 2-4-2-1 là phần bù của chính nó. Lưu ý rằng trong mã này phần bù 9 của bất kỳ số thập phân nào có thể được tính bằng cách lấy bù của các bit của các mã từ tương ứng,*

Thập phân (N)	2-4-2-1	9 – N	*Miêu tả của 9 - N trong* 2-4-2-1
0	0 0 0 0	9	1 1 1 1
1	0 0 0 1	8	1 1 1 0
2	0 0 1 0	7	1 1 0 1
3	0 0 1 1	6	1 1 0 0
4	0 1 0 0	5	1 0 1 1
5	1 0 1 1	4	0 1 0 0
6	1 1 0 0	3	0 0 1 1
7	1 1 0 1	2	0 0 1 0
8	1 1 1 0	1	0 0 0 1
9	1 1 1 1	9	0 0 0 0

1.8.3.2 *Mã theo tiêu chuẩn Mỹ đối với các thông tin trao đổi*

Máy tính cũng được sử dụng rộng rãi để xử lý các thông tin số. Để thực hiện nhiệm vụ này, phương pháp phổ biến nhất là mã hóa các ký tự đặc biệt vốn cần được xử lý. Trên một bàn phím thông thường, tiếng Anh có ít nhất là 128 ký tự khác nhau.

Các con số:	*10*
Mẫu tự (in hoa và thường):	*52*
Các ký hiệu đặc biệt:	*33 bao gồm !, @, #, $, %, ^, &, * (,) v,v...*
Các ký tự điều khiển:	*33 bao gồm Enter, space, backspace, v,v...*
	128

Số bit nhỏ nhất cần để mô tả những ký tự này là 7 do đó $2^7 = 128$ (xem phần 1.8.3). Mã 7 bit này là mã phổ biến nhất trong máy tính được biết như là mã tiêu chuẩn Mỹ cho trao đổi thông tin hoặc tên tắt

là ASCII. Tuy nhiên, bởi vì đơn vị cơ bản của việc lưu trữ là byte. Do đó, toàn bộ mã ASCII được biểu diễn bằng cách sử dụng 8 bit. Điều này cho phép hầu hết các bit biểu thị là zero. Mã khác được sử dụng rộng rãi bởi tập đoán International Business Machines là Extended Binary Coded Decimal Interchange Code hoặc EBCDIC. Trong sách này chúng ta sẽ xem xét mã ASCII. Tập hợp của các ký tự ASCII được minh họa ở chương 3. Mã ASCII được phát triển bởi viện tiêu chuẩn quốc gia Mỹ (ANSI) để cho phép trao đổi thông tin giữa thiết bị được tạo bởi nhiều nhà sản xuất khác nhau.

VÍ DỤ 1.32 *Mã ASCII mô tả thông điệp "hello World" là gì? Giả sử rằng địa chỉ đó tăng từ trái sang phải.*

Một chuỗi các ký tự được mô tả trong bộ nhớ là một chuỗi các byte liên tục. Mỗi by chứa một mã ký tự bất kỳ. Sử dụng biểu đồ trong chương 3 và giá trị thập lục phân tương đương của mỗi ký tự thông điệp này được trình bày là

H	e	l	l	o		W	o	r	l	d
48	65	6C	6C	6F	20	57	6F	72	6C	64

Nhận xét rằng ký tự trống (hex 20 hoặc decimal 32) là một ký tự giống như bất kỳ ký tự khác. Tuy nhiên, người đọc sẽ cảm thấy ký tự này rất rõ, đây là một ký tự rất khó phân biệt. Chúng ta cũng có thể giả sử rằng địa chỉ tăng từ trái sang phải để thuận tiện trong việc đọc các ký tự.

1.8.4 Dò tìm lỗi

Khi dữ liệu nhị phân được truyền lên bất cứ kiểu đường dây truyền thông nào, thì khả năng có một lỗi khi truyền luôn luôn xảy ra do hỏng thiết bị hoặc do bởi sự hiện diện của "tiếng ồn". Khi xảy ra lỗi thì có thể rằng có một hoặc nhiều bit của một byte thay đổi từ 0 đến 1 hoặc ngược lại. Bất cứ lúc nào điều này xảy ra thì một mã từ có thể thay đổi thành một mã không đúng nhưng vẫn có hiệu lực hoặc thành một chuỗi các bit không hiện diện ở bất cứ nơi nào. Trong sách này chúng ta chỉ quan tâm những dò tìm và chỉnh sửa các lỗi đơn có nghĩa rằng các lỗi mà ở đó chỉ có một là thay đổi.

Số các bit vốn thay đổi bên trong một byte trước khi byte được biến đổi sang một mã không có hiệu lực, đôi khi dùng làm một chuẩn mực để phân loại mã. Nếu chỉ có một bit của một byte cần thay đổi, thì mã này được gọi là mã tìm một lỗi. Nếu chỉ có hai bit cần phải thay đổi thì mã này được gọi là mã dò tìm hai lỗi. Để dò tìm các lỗi đơn chúng ta sử dụng một bit kiểm tra chẵn lẻ (parity check) dư. Một bit chẵn lẻ được dùng để bảo đảm rằng mỗi mã từ phải có chứa từ bit chẵn lẻ, nó có một số chẵn kiểu số 1 (gọi là được chẵn) hoặc một số lẻ cả kiểu số 1 (được lẻ). Bảng 1-6 minh họa bit chẵn ứng với mã dò tìm lỗi 8.2.4.1.

Bảng 1-6

Chữ số thập phân	Trọng lượng 8-2-4-1	Bit chẵn lẻ
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	0
6	0 1 1 0	0
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	0

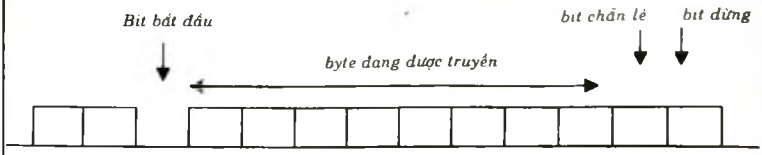
Một trong những tác vụ quan trọng nhất mà chúng ta gặp phải trong bất cứ hệ thống máy tính nào đó là việc truyền thông tin từ máy tính này sang máy tính khác. Các máy tính gửi thông tin được gọi là nguồn; các máy tính nhận thông tin được gọi là đích. Cả máy gửi và máy nhận đều tuân theo các tiêu chuẩn quốc tế để hoàn thành tác vụ này. Giả sử rằng tín hiệu được truyền được chia thành một chuỗi các đoạn 1 bít có chiều dài là T, trong suốt mỗi một đoạn này nguồn gửi hoặc một chữ 0 hoặc là một chữ số 1 (như minh họa trong hình 1-8 dưới dạng high và lows).

Để giao tiếp giữa chúng, máy tính có thể tuân theo một quy ước giống như quy ước được minh họa trong hình 1.9. Đây là trách nhiệm của đích trong việc dò tìm giá trị và tính đúng đắn của tín hiệu nhận được.

Quy trình mà hai máy tính có thể sử dụng để giao tiếp với nhau có thể được mô tả như dưới đây. Chúng ta sử dụng hình 1.9 làm ví dụ tham khảo.

(1) Nếu máy nguồn hiện không gởi bất kỳ thông báo nào, thì nó tiếp tục truyền một chuỗi các số 1.

***Hình 1-8** Biểu diễn các số 1 và 0 ở dạng cao và thấp*



Hình 1-9 *Dạng byte tiêu biểu cho sự truyền dữ liệu*

(2) Một bit 0 cho biết phần đầu của thông điệp. Bit 0 thứ nhất này được gọi là bit bắt đầu. Người ta giả định rằng các byte hình thành nên thông điệp theo sau bit bắt đầu.

(3) Bit kiểm tra tính chẵn lẻ được đặt theo qui ước về tính chẵn lẻ đang được sử dụng (chẵn hoặc lẻ)

(4) Sau bit kiểm tra tính chẵn lẻ, bit 1 khác (bit dừng) được truyền để cho biết rằng việc truyền toàn bộ các byte đã được thực hiện hoàn tất.

(5) Vào thời điểm này, máy nguồn có thể tiếp tục truyền byte kế tiếp hoặc nó có thể bắt đầu truyền một chuỗi liên tục các số 1 để chỉ ra rằng không có thêm thông điệp nào nữa được truyền vào lúc này.

VÍ DỤ 1.33 *Hai máy tính giao tiếp với nhau bằng cách sử dụng quy ước dạng dữ liệu ở hình 1-9. bộ ký tự ASCII và tính chẵn, thông điệp như minh họa dưới đây có một lỗi chẵn lẻ. Bằng cách nào máy tính nhận ra rằng một lỗi xảy ra? Máy tính có thể báo vị trí của bit bổ sung hay không? giả sử rằng bit khởi động không được minh họa. Mũi tên chỉ ra chiều mà các ký tự cần phải được xem xét.*

0101011001 0110100101 0110111101 0110110001 0110010101 0110010001
0111001111 0010000011 0110000111 0111001001 0110010101 0010000011
0110001011 0110110001 0111010111 0110010101 0010111001

Bằng cách sử dụng dạng hình 1-9 một chuỗi 10 bit, 1 byte 8 bit, 1 bit kiểm tra tính chẵn lẻ và một bit ngưng có thể được xem xét dưới dạng bằng cách trình bày một ký tự đơn của một thông điệp. Làm việc với mỗi nhóm các bit rời rạc rồi bỏ qua bit ngưng chúng ta có thể ngưng kiểm tra bit bên phải ngoài cùng bit chẵn lẻ như minh họa ở dạng in đậm dưới đây.

1.	010101100 ✔	2.	011010010 ✔	3.	011011110 ✔	4.	011011000 ✔
5.	011001010 ✔	6.	011001000 ✘	7.	011100111 ✔	8.	001000001 ✔
9.	011000011 ✔	10.	011100100 ✔	11.	011001010 ✔	12.	001000001 ✔
13.	011000101 ✔	14.	011011000 ✔	15.	011101011 ✔	16.	011001010 ✔
17.	001011100 ✔						

Bit kiểm tra chẵn lẻ cho nhóm 6, là số 1 thay vì số 0 bởi vì chúng ta đang làm việc với bit chẵn quan sát cho thấy rằng máy tính không thể báo cáo vị trí của bit bổ sung lỗi. Chúng ta sử dụng sơ đồ trong chương 3 để giải mã thông điệp. Bạn có thể dự đoán từ bit thay đổi hay không? Bạn có thể báo bit đã được bổ sung hay không? chúng ta sẽ để câu hỏi này dưới dạng một bài tập.

Ví dụ trên đây cho thấy rằng bằng cách sử dụng bit chẵn lẻ một mình máy tính có thể báo rằng có một lỗi sẽ xảy ra nhưng không thể báo vị trí của bit đã bổ sung được. Tổng quát về với n bit đã cho, để tìm một mã dò tìm lỗi không được quá một nửa của 2n tổ hợp của n bit này có thể được dùng. Bên cạnh đó các từ trong mã phải được chọn theo một cách thức như thế nào để bất cứ từ trong mã nào có thể tạo từ hiệu lực khác. Ít nhất là 2 bit phải được bổ sung. Số các bit tối thiểu cần phải được thay đổi trong một từ mã để tạo nên một từ được tạo mã đúng khác được gọi là khoảng cách tối thiểu cần phải được thay đổi trong một mã từ để tạo nên một mã từ đúng khác được gọi là khoảng cách tối thiểu của mã. Do đó chúng ta có thể thành lập lại cụm từ đã phát biểu trước đây bằng cách nói rằng với n bit đã cho, để tìm một mã dò tìm lỗi thì khoảng cách tối thiểu của nó phải là 2 hoặc lớn hơn. Mã 8.4.2.1 chẵn được minh họa trong bảng 1-6 có khoảng cách tối thiểu bằng 2.

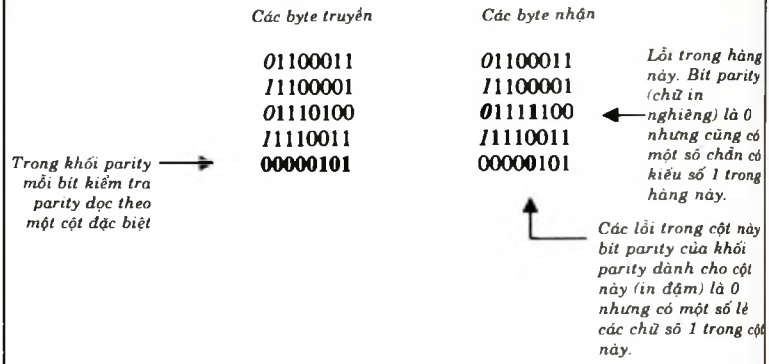
1.8.5 Chỉnh sửa lỗi

Một khi lỗi đã được tìm thấy, có một số phương pháp có thể được dùng để nhận biết và chỉnh sửa bit bổ sung. Trong phần này chúng ta sẽ xem xét một vài mã chỉnh sửa lỗi. Nói chung một mã được gọi là một mã chỉnh sửa lỗi, nếu từ của mã chỉnh sửa có thể luôn luôn xuất phát từ mã lỗi. Các phương pháp chỉnh sửa lỗi thuần (FEC) và phương pháp mã Hamming là những ví dụ của mã này. Hai phương pháp đặc biệt yêu cầu phải có các bit dư bổ sung để nhận biết và chỉnh sửa lỗi.

1.8.5.1 Chỉnh sửa lỗi thuần

Trong sơ đồ chỉnh sửa lỗi này, một khối tính chẵn hoặc lẻ bổ sung cho bit chẵn hoặc lẻ của mỗi một kiểu byte sau một số n byte được xác định sẵn. Mỗi bit trong khối chẵn hoặc lẻ kiểm tra tính chẵn lẻ của n bit đứng trước vốn chiếm giữ cùng một vị trí trong n byte trước đó. Bằng cách sử dụng thông tin ta có thể tìm và chỉnh sửa bit lỗi. Ví dụ 1.34 minh họa phương pháp này và sử dụng bit dễ nhất của một mã ASCII làm một bit kiểm tra chẵn lẻ.

***VÍ DỤ 1.34** Giả sử có hai máy tính giao tiếp với nhau bằng cách sử dụng mã ASCII mà khối kiểm tra chẵn lẻ được truyền mỗi 4 bit. Ngoài ra hãy xem xét rằng parity chẵn được dùng ở cả hai mức của byte và mức khối. Các bit parity được minh họa theo kiểu chữ in nghiêng và khối parity được minh họa theo kiểu chữ in đậm.*



Tính ra trách nhiệm của máy tính nhận đó là phải kiểm tra tính chẵn lẻ của mỗi một cột và mỗi một hàng. Bất cứ lúc nào máy tính nhận biết bit lỗi trong byte đặc biệt thì nó phải bổ sung bit để phục hồi khỏi lỗi.

1.8.5.2 Mã Hamming

Phương pháp này có thể được dùng để cung cấp việc dò tìm nhiều lỗi. Nguyên lý căn bản trong việc cấu tạo mã dành cho m bit đã cho như sau:

(1) Thêm k bit kiểm tra parity ký hiệu là p1, p2... pk và m bit đã được cho sẵn. Mã kết quả sẽ có các mã từ (m+k) bit. Chọn giá trị k để nó thỏa mãn bất đẳng thức.

$$2^k \geq m+k+1$$

Ví dụ, nếu chúng ta muốn truyền 4 bit dữ liệu (m=4) thì giá trị của k phải là 3 bởi vì

$$2^3 \geq 4+3+1$$

Kết quả này ngụ ý rằng chúng ta cần phải bổ sung 3 bit kiểm tra chẵn, lẻ ký hiệu là p1 p2 và p3.

(2) Số (m+k) bit, từ 1 cho đến m+k; bắt đầu bằng cách gán một cho bit có ý nghĩa nhất và tiếp tục cho đến khi bạn gán giá trị m+k cho bit kém ý nghĩa nhất. Phải khảo sát những số này dưới dạng vị trí của các bit trong một mã từ.

(3) Đặt các bit kiểm tra ở các vị trí 1, 2, 4... 2^{k-1} của mã từ lúc k=3 các bit kiểm tra được đặt ở vị trí 1, 2 và 2^{3-1}. Có nghĩa rằng ở vị trí 1 và 4 vị trí của các bit b_1, b_2, b_3 và b_4 và các bit kiểm tra p_1, p_2 và p_3 trong từ được tạo mã như sau. Vị trí :

Vị trí → 1 2 3 4 5 6 7
p_1 p_2 b_1 p_3 b_2 b_3 b_4

(4) *Thành lập k tập hợp, P1, P2, ... Pk của các số nhị phân. Các số nhị phân trong tập hợp Pj sẽ như thế nào để cách trình bày của chúng có k bit hoặc ít hơn và phải có một năm ở vị trí thứ j. Chọn pj theo một cách thức như thế nào để nó có parity chẵn nằm ở các vị trí được chỉ định bởi các phần tử của tập hợp parity.*

Ứng với k = 3 hãy thành lập các tập hợp P1. P2 và P3. Các phần tử trong số các tập hợp này sẽ phải có 3 bít hoặc ít hơn trong cách trình bày của chúng. Bảng 1-7 minh họa 7 tổ hợp có thể có và chúng ta có thể thành lập với ba bit thỏa mãn điều kiện của bước 4. Theo bảng này thì các tập hợp và phần tử của chúng là

P1 = {1, 3, 5, 7} P2={2, 3, 6, 7} P3={4, 5, 6, 7}

Lưu ý rằng các phần tử của P1 có các chữ số 1 trong cột 1, các phần tử của P2 phải có các chữ số 1 trong cột hai và các phần tử của P3 phải có các chữ số 1 trong cột. Do đó

chọn P1 để xác lập parity chẵn ở vị trí 1, 3 5 và 7
chọn P2 để xác lập parity chẵn ở vị trí 2, 3 6 và 7
chọn P3 để xác lập parity chẵn ở vị trí 4, 5, 6 và 7

Bảng 1-7

	P_3	P_2	P_1
1.	0	0	1
2.	0	1	0
3.	0	1	1
4.	1	0	0
5.	1	0	1
6.	1	1	0
7.	1	1	1

(5) *Dạng mã từ phải được truyền bằng cách cộng các chữ số kiểm phù hợp để điều kiện parity của các bước đây phải được thỏa mãn.*

VÍ DỤ 1.35 *Nếu dãy 1101 (dữ liệu) được truyền thì mã từ sẽ như thế nào nếu mã Hamming được dùng?*

Vị trí	1	2	3	4	5	6	7
Dữ liệu truyền	p_1	p_2	b_1	p_3	b_2	b_3	b_4
			1		1	0	1
Parity chẵn nằm ở các vị trí 1, 3, 5, 7 yêu cầu rằng p1=1	**1**		**1**		**1**	0	**1**
Parity chẵn nằm ở các vị trí 2, 3, 6, 7 yêu cầu rằng p2=0	1	**0**	**1**		1	**0**	**1**
Parity chẵn nằm ở các vị trí 5, 5, 6, 7 yêu cầu rằng p3=0	1	0	1	**0**	**1**	**0**	**1**

Mã từ được thành lập bằng cách tập trung giá trị các bit ở vị trí 1-7 ở trong trường hợp này thì từ thật sự được truyền là 1010101

Lưu ý: người đọc cần cảnh báo rằng phụ thuộc vào cách mà các bit được đánh số chúng ta có thể tìm một giá trị khác. Trong phần này như được minh họa chỉ định ở bước 2. Chúng ta gán giá trị 1 cho bit có ý nghĩa nhất và giá trị 7 cho bit có ý nghĩa nhất. Mã sẽ hoạt động với mỗi một quy ước khi người đọc thống nhất việc sử dụng sơ đồ đánh số.

Để định vị trí việc chỉnh sửa lỗi, chúng ta sử dụng các bước sau đây:

(1) Thực hành parity k kiểm tra xem các chữ số được chọn ứng với mỗi một mã từ. Kết quả của mỗi một kiểm tra parity là hoặc 0 không có lỗi xảy ra hoặc 1 nếu có một lỗi được đò tìm.

(2) Sử dụng kết quả của các thử nghiệm parity, hãy tạo nên một số nhị phân rk, r2...r1. Giá trị thập phân của số này được cho ta vị trí của chữ số bị lỗi.

VÍ DỤ 1.36 *Giả sử rằng từ được gửi là 1010101. Nếu từ được nhận là 10101101 hãy tìm vị trí của bit lỗi bằng cách sử dụng thủ tục được chỉ định trên đây.*

(1) Bởi vì ba chữ số kiểm tra đã được thêm vào. Cho nên 3 parity kiểm tra cần phải được thực hiện.

Vị trí	1	2	3	4	5	6	7	
	p_1	p_2	b_1	p_3	b_2	b_3	b_4	
Dữ liệu	1	0	1	1	1	0	1	
kiểm tra parity dành p1 =	1		1		1		1	*r1 =0 vì 0 chẵn*
kiểm tra parity dành p2 =		0	1			0	1	*r2 =0 vì 0 chẵn*
kiểm tra parity dành p3 =				1	1	0	1	*r3 = 1 vì 1 lẻ*

*Vị trí của bit lỗi là r3,r2,r1 = 100. giá trị thập phân của số này là 120100 = 1*2²=4. Kết quả này cho thấy rằng bit lỗi chiếm giữ vị trí 4 trên byte nhận đúng như giả định của chúng ta. Bít này nằm ở vị trí thứ tư sau đó được bổ sung để tạo ra thông điệp đúng.*

SOLVED PROBLEMS
Bài tập có lời giải

1.1 Early "minicomputers" had 16-bit addresses. How many different addresses did the address space have?

Since the addresses are 16 bits long, there are 2^{16} or 65,536 addresses.

1.2 Modern computers use 32-bit addresses. How many different addresses does the address space have?

Since the addresses are 32 bits long, there are 2^{32} or 4,294,967,296 unique different addresses.

1.3 Assume that a program with 4,294,836,224 bytes of instructions and data is to reside in the memory of a computer of brand XYZ. What is the minimum size (in bits) of the addresses of this computer if the program is to run successfully?

The size of the address space is given by 2^N where N is the number of bits used to represent an address. Since $2^{16} < 4{,}294{,}836{,}224 < 2^{32}$, the minimum number of bits necessary to represent addresses is 32.

14 The ASCII representation (in hexadecimal) of the character string "HI THERE!" is shown below. If the H is stored in byte 5C1B, what is the address of the byte that contains the letter R? Would the address of the byte that contains the letter R be different if all the letters were lowercase? To facilitate the reading of the string we will assume that the addresses increase toward the right.

H	I		T	H	E	R	E	!
48	49	20	54	48	65	52	65	21

5C1B

The byte that contains the letter R is 6 bytes away, therefore, its address is

```
  5C1B +
     6
  ------
  5C21
```

The address of this byte remains the same regardless of its contents.

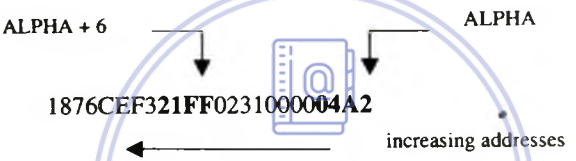
1.5 The hard disk of a computer has a capacity of 4 Gigabytes; how many bytes does it really have?

4 Gigabytes = $4*10^6$ Kilobytes = $4*10^6*1024$ bytes = 4,096,000,000 bytes

1.6 Some instructions like the one shown below explicitly state the size of their operands. In this case the instruction "moves" (copies) the content of the word beginning at location ALPHA to the word beginning at location ALPHA + 6. Using the instruction indicated below, show the contents of these memory locations after the instruction gets executed. Assume that a word occupies two bytes.

MOVW ALPHA, ALPHA + 6

Content of memory before the instruction gets executed (shown in hexadecimal).



ALPHA is a named memory location or a symbolic address. The address ALPHA + 6 is six bytes away from ALPHA in the direction in which the addresses increase. Since a word occupies two bytes, the word beginning at ALPHA has A242 as its content. These two bytes are copied into position ALPHA + 6, therefore, the contents of these memory locations after the instruction is executed looks like this:

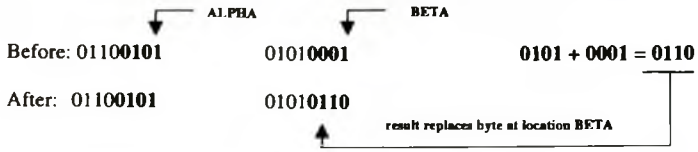
1876CEF304A20231000004A2

Notice that the content of location ALPHA has not changed.

1.7 Show the contents of locations ALPHA and BETA after executing the instruction indicated below. Assume that the initial contents of the operands at locations ALPHA and BETA are 01100101 and 01010001 respectively. Do all operations in two's complement arithmetic.

ADD2B ALPHA, BETA

In this case the instruction calls for adding the bytes at location ALPHA and BETA. This situation is pictured below with the operands shown in bold before and after the execution of the operation.



As already mentioned, the operator calls for adding two bytes. The symbolic addresses of these two bytes are ALPHA and BETA respectively. Since there are two operands the addition is destructive (see Example 1.4). Therefore, the byte at location BETA will be replaced by the result of the operation. This example also illustrates that the address of any memory location is the address of its first byte in the direction in which the addresses increase.

1.8 Given the sequence $10001100_{(2)}$, show its decimal equivalent if it is considered (a) an unsigned binary number, (b) a sign-magnitude number, (c) a two's complement number.

(a) If this sequence is considered as an unsigned binary number its decimal equivalent is

$1^7 0^6 0^5 0^4 1^3 1^2 0^1 0^0 = (1*2^7) + (1*2^3) + (1*2^2) = 128 + 8 + 4 = 140$

(b) If this sequence is considered as a sign-magnitude number we have that

Sign bit = 1. Therefore, the number is negative.

Magnitude: $0^6 0^5 0^4 1^3 1^2 1^0 0^0 = (1*2^3) + (1*2^2) = 8 + 4 = 12$

Therefore, the decimal equivalent of this number is -12.

(c) If this sequence is considered as a two's complement number we have that

Sign bit = 1. Therefore, the number is negative.

Using the procedure described in Section 1.8.2 we need to

(1) complement of the number: 10001100 → 01110011 (Change 0's to 1's and 1's to 0's)

(2) add one to the complemented number

$$\begin{array}{r} 01110011 + \\ \quad\quad 1 \\ \hline 01110100 \end{array}$$

The decimal equivalent of this last sequence is $0^7 1^6 1^5 1^4 0^3 1^2 0^1 0^0$ = $(1*2^6) + (1*2^5) + (1*2^4) + (1*2^2)$ = 116. Therefore, the decimal equivalent of the number is -116.

1.9 What is the decimal equivalent of $CA14_{(16}$?

Using the procedure of Section 1.4.2 we have that

$$CA14_{(16} = C*16^3 + A*16^2 + 1*16^1 + 4*16^0$$

Replacing C for 12 and A for 10, their respective decimal equivalents, we obtain

$CA14_{(16}$ = (12*4096) + (10*256) + (1*16) + (4*1)

= 51,732

1.10 What is the hexadecimal equivalent of decimal 331?

Since we want to convert a decimal number to hexadecimal we need to divide the number by the hexadecimal basis. That is, by 16. Using the procedure of Section 1.5.2 we have that

Number	Quotient When Dividing by 16	Remainder
331	20	11 (=B)
20	1	4
1	0	1

The hexadecimal equivalent of decimal 331 is $14B_{(16}$.

We can verify that this is the correct result as follows:

$$14B_{(16} = (1*16^2) + (4*16^1) + (B*16^0)$$

Replacing B by its decimal equivalent we have that

$$14B_{(16} = (1*256) + (4*16) + (11*1)$$
$$= 256 + 64 + 11$$
$$= 331$$

1.11 What is the binary equivalent of decimal 115?

Since we want to convert a decimal number to binary we need to divide the number by the binary basis. That is, by 2. To obtain the binary equivalent follow the procedure of Section 1.5.2.

Number	Quotient When Dividing by 2	Remainder
115	57	1
57	28	1
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1

The binary equivalent of decimal 115 is $1110011_{(2}$.

We can verify that the result is correct by noting that

$$1^61^51^40^30^21^11^0{}_{(2} = (1*2^6) + (1*2^5) + (1*2^4) + (0*2^3) + (0*2^2) + (1*2^1) + (1*2^0)$$
$$= (1*64) + (1*32) + (1*16) + (1*2) + (1*1)$$
$$= 64 + 32 + 16 + 2 + 1 = 115$$

Another way of obtaining the binary equivalent of the decimal number is to convert this value to its hexadecimal or octal equivalent and then express it in binary. This is illustrated below.

Number	Quotient When Dividing by 16	Remainder
115	7	3
7	0	7

The equivalent hexadecimal number of decimal 115 is 73(16. Expressing this hexadecimal number in binary according to the procedure of Section 1.5.1 we have that

$$73_{(16} = 01110011_{(2}$$

1.12 What is the octal equivalent of decimal 1144?

Since we want to convert a decimal number to an octal we need to divide the number by the octal basis. That is, by 8. Using the procedure of Section 1.5.2 we have that

Number	Quotient When Dividing by 8	Remainder
1144	143	0
143	17	7
17	2	1
2	0	2

The octal equivalent of decimal 1144 is $2170_{(8}$. We can verify this result as follows:

$$\begin{aligned} 2170_{(8} &= (2*8^3) + (1*8^2) + (7*8^1) + (0*8^0) \\ &= (2*512) + (1*64) + (56) + (0*1) \\ &= 1024 + 64 + 56 \\ &= 1144 \end{aligned}$$

1.13 What is the octal equivalent of $1010011101012_{(2}$?

Since 23 = 2^3, we can write each individual octal number using only three bits. We can use a procedure similar to the one used for converting binary numbers to their hexadecimal equivalents (see Section 1.5.1). However, instead of forming groups of four bits we form groups of three bits. Therefore,

$$1010011101012_{(2} = 101\ 001\ 110\ 101 = 5165_{(8}$$

1.14 What is the hexadecimal equivalent of 111101001102?

Forming groups of four bits beginning from the rightmost bit according to the procedure of Section 1.5.1 we have that

$$11110100112 = 1\ 1110\ 1001 = \mathbf{000}1\ 1110\ 1001$$

Notice that it was necessary to add three bits (shown in bold) to the left of the single bit of the leftmost group to make it a four-bit group.

Using Table 1-3, the hexadecimal equivalent of the given binary number is

$$111101001_{(2} = 1\ 1110\ 1001 = \mathbf{000}1\ 1110\ 1001 = 1E9_{(16}$$

1.15 What is the result of adding the two's complement numbers 01100100 and 00011100? If an overflow occurs explain why.

$$\begin{array}{l} 01100\mathbf{100} + \\ 00011\mathbf{100} \\ \hline \mathbf{00} \end{array}$$

← zero plus zero is zero. Therefore, write zero.

```
01100100 +
00011100
--------
```

000 ← One plus one is zero with a carry of one. Therefore, write zero and carry one.

```
  1
01100100 +
00011100
--------
```

0000 ← One (the carry) plus zero is one. This one plus one is zero with a carry of one. Therefore, write zero and carry one.

```
  1
01100100 +
00011100
--------
```

000000 ← One (the carry) plus one is zero with a carry of one. This zero plus zero is zero. Therefore, write zero and carry one.

```
 1
01100100 +
00011100
--------
```

0000000 ← One (the carry) plus one is zero with a carry of one. This zero plus zero is zero. Therefore, write zero and carry one.

```
1
01100100 +
00011100
--------
```

10000000 ← One (the carry) plus zero is one. This one plus zero is one. Therefore, write one.

Notice that both operands are positive but the result is negative. Therefore, the result overflows.

The result of this addition illustrates another way of determining if an overflow has occurred. Notice that there was a carry into the sign bit but there was no carry out of it.

1.16 What is the result of adding the two's complement numbers 10011011 and 11100011? If an overflow occurs explain why.

```
10011011 +
11100011
--------
```

0 ← One plus one is zero with a carry of one. Therefore, write zero and carry one.

```
        1
10011011 +
11100011
--------
```

10← One (the carry) plus one is zero with a carry of one. This zero plus one is one. Therefore, write one and carry one.

```
      1
10011011 +
11100011
--------
```

110← One (the carry) plus zero is one. This one plus zero is one. Therefore, write one. 10011011 + 11100011

```
10011011 +
11100011
--------
```

1110← One plus zero is one. Therefore, write one

```
10011011 +
11100011
--------
```

11110← One plus zero is one. Therefore, write one.

```
10011011 +
11100011
--------
```

111110← Zero plus one is one. Therefore, write one.

```
10011011 +
11100011
--------
```

1111110← Zero plus one is one. Therefore, write one.

```
Carry out ← 1 10011011 +
              11100011
              --------
```

01111110← One plus one is zero with a carry of one. Therefore, write zero and carry one.

The addition overflows since the operands are both negative and the result is positive. This result illustrates another condition for detecting an overflow. Notice that there is a carry out of the sign bit but there is no carry into it.

This condition and the one already mentioned in solved problem 1.15 allows us to detect when an overflow has occurred. Therefore,

we can say that when performing additions in two's complement an overflow occurs if

(a) there is a carry into the sign bit and no carry out of the sign bit

or

(b) there is a carry out of the sign bit and no carry into the sign bit.

1.17 What is the result of adding the hexadecimal numbers 143AF and 215137?

```
  143AF +
  215B7
 --------
      6   ← "Thinking in decimal" we have that 15 plus 7 is 22. Since 22 = 1*16 + 6, write 6 and carry 1.

    1
  143AF +
  215B7
 --------
     66   ← 1 (the carry) plus ten is eleven. 11 plus 11 is 22. Since 22 = 1*16 + 6, write 6 and carry 1.

    1
  143AF +
  215B7
 --------
    966   ← 1 (the carry) plus three is four. 4 plus 5 is 9. Therefore write 9.

  143AF +
  215B7
 --------
   5966   ← Four plus one is five. Therefore write 5.

  143AF +
  215B7
 --------
  35966   ← One plus two is three. Therefore write 3.
```


1.18 What is the result of multiplying 00001101 by 00000101?

Multiplication in binary can be carried out the same way we multiply in decimal. The major difference is that we are restricted to the use of the symbols 0 and 1. In this case $1101_{(2}$ = 13 and $101_{(2}$ = 5.

```
1101 ×
 101
------
1101 +
0000
1101
------
1000001
```

Notice that $1000001_{(2}$ = 65. This is the same result that we would have obtained in the decimal system since 65 = 13*5.

1.19 What is the decimal equivalent of the unsigned binary number 11001001?

Using the procedure indicated in Section 1.1.1 we have that

$$\begin{aligned} 11001001_{(2} &= 1^{7}1^{6}0^{5}0^{4}1^{3}0^{2}0^{1}1^{0} \\ &= (1*2^{7}) + (1*2^{6}) + (1*2^{3}) + (1*2^{0}) \\ &= 128 + 64 + 8 + 1 \\ &= 201 \end{aligned}$$

Another method that is widely used for calculating the decimal equivalent of a binary number is Horner's method. This procedure, which works for any base, is as follows:

"Start with the first digit on the left and multiply it by the base. Then add the next digit and multiply the sum by the base. Continue this process until you add the last digit."

In this particular case we have to multiply by 2 since the number is binary.

$11001001_{(2} \rightarrow 1*2 = 2$
$2 + 1 = 3$ and $3*2 = 6$
$6 + 0 = 6$ and $6*2 = 12$
$12 + 0 = 12$ and $12*2 = 24$
$24 + 1 = 25$ and $25*2 = 50$
$50 + 0 = 50$ and $50*2 = 100$
$100 + 0 = 100$ and $100*2 = 200$
$200 + 1 = 201$

1.20 Use Homer's method to calculate the decimal equivalent of the number 2341(s.

Since the number is octal we need to multiply by 8.

$231_{(8} \rightarrow 2*8 = 16$

$16 + 3 = 19$ and $19*8 = 152$

$152 + 4 = 156$ and $156*8 = 1248$

$1248 + 1 = 1249$.

Therefore, $2341_{(8} = 1249$

1.21 Another convention for representing both positive and negative numbers in a computer is the **one's complement notation or 1's complement.** This convention was devised to make the addition of two numbers with different signs the same as for two numbers with the same sign. In this convention, positive numbers are represented in the usual way. To represent a negative number start with the binary representation of the absolute value of the number and complement all its bits. To carry out arithmetic operations treat the sign bit as any other bit. However, when performing an addition, if there is a carry out of the most significant bit add this bit to the rightmost bit. This is known as the ***end-around carry.***

Overflows are detected following similar conventions to those of the 2's complement (see Section 1.8.2.2 and solved problems 1.15 and 1.16). The range of values that can be represented in this convention using n bits is -2^{+n} to $+2^n - 1$.

What is the 1's complement representation of decimal -35? Use a byte to represent the number.

(1) Start with the binary representation of the absolute value of the number. In this case,

$$35 = 00100011_{(2}$$

(2) Complement all the bits of the binary number obtained in the previous step.

00100011

Changing 0's to 1's and vice versa

11011100

The 1's complement of decimal -35 is 11011100.

1.22 Given the bit patterns shown below, find their equivalent values in the conventions indicated below.

Bit Pattern	Sign-magnitude	1's Complement	2's Complement	Unsigned
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	−0	−3	−4	4
101	−1	−2	−3	5
110	−2	−1	−2	6
111	−3	−0	−1	7

This example shows that in order to interpret the content of any memory location correctly it is necessary to know the convention being used. Notice that the positive values have the same representation in all conventions. The zero value has two different representations in sign-magnitude and 1's complement. This complicates the logic of the ALU.

1.23 What is the result of adding the 1's complement numbers 00100001 and 11101010?

00100001 +
11101010
————————
00001011 — One plus zero is one.
00100001 +
11101010
————————
00001011 — Zero plus one is one.
00100001 +
11101010
————————
00001011 — Zero plus zero is zero.
00100001 +
11101010
————————
00001011 — Zero plus one is one.
00100001 +
11101010
————————
00001011 — Zero plus zero is zero.
00100001 +
11101010
————————
00001011 — One plus one is zero with a carry of one.

00100001 +
11101010
———————

00001011 ← One plus zero is one.
00100001 +
11101010
———————

00001011 ← Zero plus one is one.
00100001 +
11101010
———————

00001011 ← Zero plus zero is zero.
00100001 +
11101010
———————

00001011 ← Zero plus one is one.
00100001 +
11101010
———————

00001011 ← Zero plus zero is zero.
00100001 +
11101010
———————

00001011 ← One plus one is zero with a carry of one.

1.24 What is the result of 10001011 -10101 in two's complement?

10001011 −
10101
———————

0 ← One minus one is zero.
10001011 −
10101
———————

10 ← One minus zero is one.
10001011 −
10101
———————

110 ← Zero is less than one. Borrow "two" from higher unit. Two plus zero is two. Two minus one is one.
0
10001011 −
10101
———————

0110 ← The one (in bold) became zero since it "paid" the "borrowed" two. Zero minus zero is zero.

```
10001011 –
   10101
```

10110 — Zero is less than one. Borrow "two" from higher unit. Two plus zero is two. Two minus one is one.

```
   1
10001011 –
   10101
```

110110 — Zero became one (basis minus one). One minus zero (not shown) is one.

```
  1
10001011 –
   10101
```

1110110 — Zero became one (basis minus one). One minus zero (not shown) is one.

```
 0
10001011 –
   10101
```

01110110 — Zero became one (basis minus one). Zero minus zero (not shown) is zero.

1.25 What is the result of the following hexadecimal operation: 9A8C7B - BD35?

```
9A8C7B –
   BD35
```

99CF46 — Eleven minus five is six.

```
9A8C7B –
   BD35
```

99CF46 — Seven minus three is four.

```
9A8C7B –
   BD35
```

99CF46 — Twelve is less than thirteen. Borrow "16" from higher unit. Sixteen plus twelve is twenty-eight. Twenty-eight minus thirteen is fifteen. Therefore, write F.

```
   7
9A8C7B –
   BD35
```

99CF46 — Eight became seven since it paid the "borrowed" sixteen. Seven is less than eleven. Borrow sixteen from higher unit. Seven plus sixteen is twenty-three. Twenty-three minus eleven is twelve.

```
 9
9A8C7B –
   BD35
```

99CF46 — The A (ten) became nine since it paid the "borrowed" sixteen. Nine minus zero (not shown) is nine.

1.26 If n bits can represent 2^n different numbers, why is the largest unsigned number only 2^{n-1} and not 2^n?

There are 2^n different numbers between 0 and 2^{n-1}. However, since the sequence begins with 0 the largest representation is 2^{n-1}.

1.27 The error detection code shown below is known as the "2-out-of-five" code since it consists of all possible combinations of two 1's in a five-bit coded word. What is the value of the parity check bit if even parity is assumed?

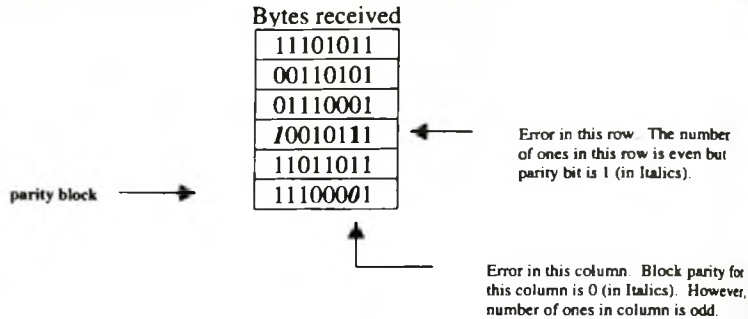
Decimal Value	Weight 0 1 2 4 7
0	0 0 0 1 1
1	1 1 0 0 0
2	1 0 1 0 0
3	0 1 1 0 0
4	1 0 0 1 0
5	0 1 0 1 0
6	0 0 1 1 0
7	1 0 0 0 1
8	0 1 0 0 1
9	0 0 1 0 1

The number of bits in every row is even, therefore the parity bit is zero in all rows.

Note: in this code, with the exception of the code word for decimal 0, all entries can be derived from the 1-2-4-7 code. This also shows that the assignment of code word to decimal digits does not necessarily follow a logical pattern.

1.28 Using the Forward Error Code of Section 1.8.5.1 find the bit in error for the sequence of bytes shown below. Assume that (a) a parity block is sent every five bytes and that even parity is being used at both the byte and the block level, (b) the leftmost bit of every byte is used as parity bit.

	Bytes sent
	11101011
	00110101
	01110001
	10010101
	11011011
parity block →	11100001



1.29 Use the format of Fig. 1-9 to find the error in the following sequence of bytes. Assume even parity.

1010111011 0110100101 0110111101 0110110001 0110010101 0110010001
0111001111 0010000011 0110000111 0111001001 0110010101 0010000011
0110001011 0110110001 0111010111 0110010101 0010111001

Working with each group separately and dropping the stop bit we can check the rightmost bit (shown here in bold).

1. 101011101 ✔	2. 011010010 ✔	3. 011011110 ✔	4. 011011000 ✔
5. 011001010 ✔	6. 011001001 ✔	7. 011100111 ✔	8. 001000001 ✔
9. 011000011 ✔	10. 011100100 ✔	11. 011001010 ✔	12. 001000001 ✔
13. 011000101 ✔	14. 011011000 ✔	15. 011101011 ✔	16. 011001011 ✘
17. 001011100 ✔			

The parity bit of group No. 16 should be zero instead of one.

1.30 Using the Hamming code method of Section 1.8.5.2 and four-bit data, what are the values of the checking bits if the message to be sent is 0110? Show the procedure to determine that an error has occurred in the 6th position of the received message.

Position	1	2	3	4	5	6	7
Parity checking bits	p_1	p_2	b_1	p_3	b_2	b_3	b_4
Data			0		1	1	0
p_1 requires even parity in positions 1, 3, 5, 7	1		0		1		0
p_2 requires even parity in positions 2, 3, 6, 7		1	0			1	0
p_3 requires even parity in positions 4, 5, 6, 7				0	1	1	0
word to be transmitted	1	1	0	0	1	1	0

If an error has occurred in the 6th position it is because the message 1100101 was received. To determine the position of the erroneous bit the destination needs to perform the following three parity checks.

Position	1	2	3	4	5	6	7
Parity checking bits	p_1	p_2	b_1	p_3	b_2	b_3	b_4
Data received	1	1	0	0	1	0	0
$r_1 = 0$ since parity is even in pos. 1, 3, 5, 7	1		0		1		0
$r_2 = 1$ since parity is even in pos. 2, 3, 6, 7		1	0			0	0
$r_3 = 1$ since parity is even in pos. 4, 5, 6, 7				0	1	0	0

To find the position of the erroneous bit concatenate the results of the parity bits in the order $r_3r_2r_1$. The decimal equivalent of the binary number 110 gives the position of the erroneous bit. The erroneous bit is in position 6 as it should be.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

1.1 *Các máy tính mi ni trước đây có 6 bít địa chỉ. Hỏi có bao nhiêu địa chỉ khác nhau mà không gian địa chỉ có được?*

Bởi vì các địa chỉ dài 16 bít cho nên có 216 hoặc 65,536 địa chỉ

1.2 *Các máy tính hiện đại sử dụng các địa chỉ 32 bít. Hỏi có bao nhiêu địa chỉ khác nhau mà khoảng trống địa chỉ có được?*

Bởi vì các địa chỉ dài 32 bít cho nên có 232 hoặc 4,294,967,296 địa chỉ khác nhau.

1.3 *Giả sử rằng một chương trình có 4,294,836,224 byte lệnh và dữ liệu nằm bên trong bộ nhớ của máy tính mang nhãn là XYZ. Hỏi kích thước tối thiểu (tính theo bít) của các địa chỉ trong máy tính này bằng bao nhiêu. Nếu chương trình này chạy thành công?*

1.4 *Các trình bày ASCII (theo hệ 16) của chuỗi ký tự "HI THERE" được minh họa dưới đây. Nếu H được lưu trữ trong byte 5C1B, thì địa chỉ của byte có chứa mẫu tự R là như thế nào? địa chỉ của byte có chứa mẫu tự có khác hay không nếu tất cả mẫu tự được viết theo kiểu chữ thường? Để đọc được chuỗi này ta sẽ giả sử rằng địa chỉ tăng hướng về phía bên phải.*

1.5 *Đĩa cứng của một máy tính có dung lượng là 4 gigabyte. Hỏi nó thật sự có bao nhiêu byte?*

1.6 *Một vài lệnh giống như lệnh được minh họa dưới đây trình bày trạng thái kích thước của các toán hạng. Trong trường hợp lệnh này "di chuyển" (sao chép) nội dung của từ bắt đầu ở vị trí ALPHA sang từ bắt đầu ở vị trí ALPHA + 6. Bằng cách sử dụng lệnh chỉ định dưới đây, hãy biểu thị nội dung của các vị trí bộ nhớ này sau khi lệnh được thực thi. Giả sử rằng có một từ chiếm hai byte.*

MOVW ALPHA ALPHA + 6

1.7 *Biểu thị nội dung của các vị trí ALPHA và BETA sau khi thực thi lệnh được chỉ định dưới đây. Giả sử rằng nội dung khởi tạo của các toán hạng nằm tại các vị trí ALPHA và BETA là 01100101 và 01010001 tương ứng. Thực hiện tất cả các phép tính trong số học phần bù của hai.*

1.8 *Cho dãy tuần tự $10001100_{(2}$, hãy biểu thị tương đương thập phân của nó, nếu nó được xem như (a) một số nhị phân không có dấu (b) một số có dấu và độ lớn (c) một số phần bù của hai.*

1.9 *Tương đương thập phân của $CA14_{(16}$ bằng bao nhiêu?*

1.10 *Tương đương thập lục phân của số thập phân 331 bằng bao nhiêu?*

1.11 *Tương đương nhị phân của số thập phân 115 bằng bao nhiêu?*

1.12 *Tương đương bát phân (hệ 8) của số thập phân 1144 bằng bao nhiêu?*

1.13 *Tương đương bát phân (hệ 8) của $101001110101_{(2}$ bằng bao nhiêu?*

1.14 *Tương đương thập lục phân (hệ 16) của $111101001_{(2}$ bằng bao nhiêu?*

1.15 *Tìm kết quả của phép cộng các số bù của hai 01100100 và 00011100 bằng bao nhiêu? Nếu có một sự quá tải xảy ra hãy giải thích lý do?*

1.16 *Tìm kết quả của phép cộng các số bù của hai 10011011 và 11100011 bằng bao nhiêu? Nếu có một sự quá tải (quá dòng) xảy ra hãy giải thích tại sao?*

1.17 *Tìm kết quả của phép cộng các số thập lục phân (hệ 16) 143AF và 215B7?*

1.18 *Tìm kết quả của phép nhân 00001101 cho 00000101?*

1.19 *Tìm tương đương thập phân của các số nhị phân không có dấu 11001001?*

1.20 *Sử dụng phương pháp Horner để tính tương đương thập phân của số $2341_{(8}$*

1.21 *Có một quy ước khác được trình bày các số dương và số âm trong một máy tính đó là sử dụng ghi chú phần bù của một*

khoảng phần bù của số 1. Quyển này được rút ra để thực hiện phép cộng hai số có dấu khác nhau giống hệt như phép cộng hai số có cùng dấu. Theo quy ước này các số dương được trình bày theo cách thức thông thường. Để trình bày một số âm bắt đầu bằng một cách trình bày nhị phân có giá trị tuyệt đối của số đó và phần bù của tất cả bit của nó. Để thực hiện các phép tính số học ta hãy xử lý bit dấu dưới dạng bất cứ bit nào khác. Tuy nhiên, lúc thực hiện một phép cộng, nếu có thực thi bit có ý nghĩa nhất thì hãy cộng bit này cho bit bên phải ngoài cùng (bit phải nhất). Phương pháp này được gọi là phép mang vòng tận cùng (end-around-cary).

Sự quá dòng được dò tìm tuân theo các quy ước tương tự với những quy ước phần bù của hai (xem 1.8.2.2 và bài tập có lời giải 1.15 và 1.16). Miền các giá trị có thể được trình bày theo quy ước này bằng cách sử dụng n bit là –2n-1 cho đến +2n-1 – 1. Hãy trình bày phần bù của 1 đối với số thập phân –35? Hãy sử dụng byte để trình bày số này.

1.22 *Cho các mẫu bit như minh họa dưới đây. Hãy tìm giá trị tương đương theo những quy ước được chỉ định bên dưới.*

1.23 *Kết quả của việc cộng các số bù 1 của 00100001 và 111011010 bằng bao nhiêu?*

1.24 *Kết quả của phép trừ 10001011 – 10101 theo bù 2 bằng bao nhiêu?*

1.25 *Kết quả của phép tính thập lục phân (hệ 16) 9A8C7B – BD35?*

1.26 *Nếu các n bit có thể biểu thị 2^n khác nhau, thì tại sao số không có dấu lớn nhất chỉ là 2^{n-1} chứ không phải là 2^n?*

1.27 *Mã tìm lỗi được minh họa dưới đây được gọi là mã "5 chọn lấy 2" bởi vì nó bao gồm tất cả tổ hợp của hai chữ số 1 trong một từ được tạo mã 5 bit. Tìm giá trị của bit kiểm tra tính chẵn lẻ. Nếu giả sử ta có parity chẵn.*

1.28 *Sử dụng mã tìm lỗi thuận (Forward Error Code) của 1.8.5.1 hãy tìm bit lỗi cho chuỗi các byte được minh họa dưới đây. Giả sử rằng (a) một khối parity gửi mỗi 5 byte và rằng parity chẳng được dùng ở cả mức byte lẫn mức khối (b) bit bên trái ngoài cùng của mỗi một byte được dùng làm bit parity.*

1.29 *Sử dụng dạng ở hình 1-9 hãy tìm lỗi trong chuỗi sau đây. Giả sử rằng parity chẵn*

1.30 *Sử dụng phương pháp tạo mã Hamming của 1.8.5.2 và dữ liệu 4 bit, tìm giá trị của các bit kiểm tra nếu thông điệp được gửi đến là 0110? Biểu thị thủ tục để xác định rằng có một lỗi xảy ra ở vị trí thứ 6 của thông điệp nhận.*

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

1.31 *Trong hình 1-3 một địa chỉ sau cùng được cho dưới dạng N-1. Tại sao có điều này? Nếu một vị trí bộ nhớ (1 byte) có một địa chỉ là 100 thì có bao nhiêu byte đứng trước nó?*

1.32 *Nếu một máy có địa chỉ 4 bit, thì kích thước của không gian địa chỉ bằng bao nhiêu? Địa chỉ của vị trí bộ nhớ sau cùng bằng bao nhiêu?*

1.33 *Một chương trình mà dữ liệu của nó dài 63,816 byte. Số các bit tối thiểu cần thiết để trình bày các địa chỉ bằng bao nhiêu nếu chương trình dữ liệu của nó chuẩn bị đưa vào bên trong bộ nhớ của một lúc.*

1.34 *Giả sử rằng máy XYZ sử dụng các toán hạng byte trong tất cả các lệnh của nó và rằng máy WRT sử dụng các toán hạng từ trong tất cả các lệnh của nó. Nếu cả hai máy này có địa chỉ 32 bit, thì máy nào trong số hai máy có vùng không gian địa chỉ lớn nhất.*

1.35 *Biểu thị nội dung của các vị trí ALPHA và BETA sau khi thực thi lệnh được chỉ định dưới đây. Giả sử rằng nội dung khởi tạo của toán hạng từ nằm ở vị trí ALPHA và BETA là 00101010 và 00001100 tương ứng. Hãy thực hiện tất cả phép tính trong số học bù của hai.*

1.36 *Giả sử rằng nội dung vị trí của ALPHA và BETA sau khi thực thi lệnh được chỉ định dưới đây. Giả sử nội dung ban đầu của các toán hạng từ nằm ở vị trí ALPHA và BETA là 01100101 và 01010010 tương ứng. Hãy thực hiện tất cả các phép tính số học bù của hai.*

1.37 *Cho một số nhị phân, bằng cách nào chúng ta biết được giá trị tương đương của thập phân của nó là lẻ hoặc chẵn?*

1.38 *Giả sử rằng trong suốt quá trình truyền 1 byte có hai trong số bit của nó được bù nếu parity chẵn được dùng, thì máy tính có thể dò tìm một lỗi có thể xảy ra trong byte đó được không?*

1.39 *Các số thập lục phân ASCII nào có thể được dùng dưới dạng các từ tiếng Anh?*

1.40 *Thế nào là 10? Gợi ý: giả sử rằng 10 được viết dưới dạng nhị phân, bát phân và thập lục phân và tìm tương đương thập phân của nó.*

1.41 *Bằng cách nào chúng ta xác định rằng một hiện tượng quá dòng là xảy ra lúc thực thi một phép cộng với các số nhị phân không có dấu? Giả sử rằng các toán hạng đều là các số n bit.*

1.42 *Phần bù của hai là quy ước phổ biến nhất để trình bày cả số dương lẫn số âm bởi vì nó không có bất kỳ bài tập nào của phần bù 1 hoặc dấu – độ lớn. Những bài tập tuân theo hai quy ước này là như thế nào?*

1.43 *Hãy hoàn tất bảng sau đây bằng cách biến đổi các số được cho trong mỗi một hàng thành các cơ số tương ứng khác.*

1.44 *Kết quả của các phép tính số học thập lục phân sau đây thì bằng bao nhiêu?*

1.45 *Tương đương thập phân của chuỗi bit 10101010 bằng bao nhiêu nếu nó được trình bày theo các quy ước được chỉ định dưới đây? Giả sử rằng byte là đơn vị cơ bản.*

1.46 *Kết quả của phép cộng số bù 2 được minh họa dưới đây bằng bao nhiêu? Trong mỗi trường hợp hãy chỉ định cho biết có xảy ra hiện tượng quá dòng hay không?*

1.47 *Kết quả của phép trừ số bù của 2 bằng bao nhiêu?*

ANSWERS TO SUPPLEMENTARY PROBLEMS
Trả lời các bài tập bổ sung

1.31 Addresses begin with 0, therefore, the last address is always N - 1. There are 100 addresses between 0 and 99. If we include the 100th location, there are 101 memory locations.

1.32 The size of the address space is 2'. The address of the last memory location is 2' -1.

1.33 The minimum address size is 2^{16} since $2^{15} < 63816 < 2_{16}$.

1.34 The address space of both machines is 2'2 bytes long.

1.35 After executing the instruction, the content of location BETA is 00110110. Location ALPHA remains unchanged.

1.36 After executing the instruction, the content of location BETA is 01010111. Notice that only the byte at location BETA was changed. Location ALPHA remains unchanged.

1.37 Any binary number with an odd decimal equivalent has its rightmost bit equal to 1. Likewise, any binary number with an even decimal equivalent has its rightmost bit equal to 0.

1.38 No, if two of the bits are complemented the number of 1-bits and 0-bits remains the same. Therefore, the computer cannot tell that an error has occurred.

1.39 All ASCII codes within the hexadecimal range 41 through 5A represent uppercase letters. Hexadecimal ASCII values 61 through 7A represent lowercase letters.

1.40 Regardless of the value of the basis r (r > 0), in all numerical positional systems, 10 always represents the value of the basis.

1.41 The result of the addition will have n + 1 bits.

1.42 Problems with sign-magnitude: arithmetic operations with numbers of different signs increase the complexity of the ALU. Adding two numbers with different signs requires that the sign and magnitude of the larger number be determined before the addition is carried out. This increases the overhead considerably, particularly if there are millions of operations to be performed. Problems with 1's complement: there are two representations for zero that add unnecessary testswhen performing arithmetic operations. The carry around bit increases the amount of work required to add two numbers.

1.43

Decimal	Binary (unsigned)	Octal	Hexadecimal
125	1111101	175	7D
137	10001001	211	89
214	11010110	326	D6
14895	11101000101111	35057	3A2F
1949	11110011101	3635	79D

1.44 (a) BE3B *(b)* 8CBA (c) 9F2F *(d) 11131*

1.45 (a) 170 (unsigned) *(b)* -34 (sign-magnitude) (c) -85 (1's complement) *(d)* -86 (2's complement)

1.46 (a) 00111111 *(b) 10111000 (c) 10101010 (d) 11111001*

1.47 (a) 01010010 *(b) 11011101 (c) 11000111 (d) 10111100*

CHAPTER 2

Program Planning and Design

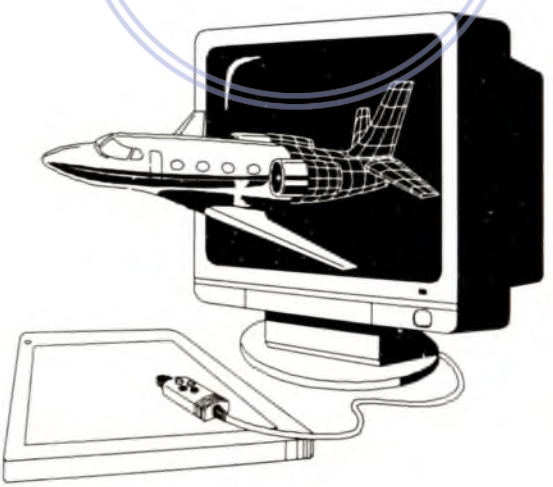
Thiết kế và hoạch định chương trình

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các vấn đề liên quan đến việc thiết kế và hoạch định chương trình, cụ thể với các nội dung cơ bản sau đây:

- Programming
- Problem solving
- Algorithms
- *Lập trình*
- *Giải quyết vấn đề*
- *Các thuật toán*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.



CHỦ ĐIỂM 2.1

PROGRAMMING
Lập trình

The basic flow of computer processing, as shown in Fig. 2-1, is "Input, Process, Output." The art of **programming** involves writing instructions to tell the computer how to process specific information. Some kind of data is sent into the program, and then the processed data are sent out from the program. For example, in a payroll-processing program, the hours worked and hourly salary for a number of employees might be the input. The program would then calculate the gross pay, the various deductions to be subtracted, and the net pay. One output would be the actual paychecks. In order for the processing to be completed successfully, the program must execute the correct calculations in the correct order.

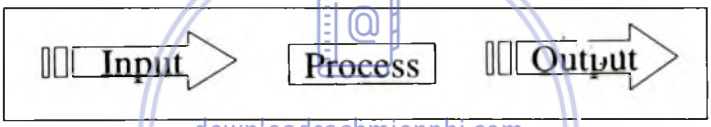


Fig. 2-1 Processing flow.

EXAMPLE 2.1 What would be the input, process, and output for a program to calculate a semester grade point average?

Input: Letter grades from all the courses in the semester, and the grading scale used (e.g. A = 4.0, B = 3.0, etc.)

Process: Add up the numerical values of all the course grades and divide by the number of courses

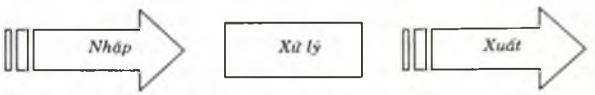
Output: The grade point average for the semester

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 2.1

2.1 LẬP TRÌNH

Một dòng xử lý máy tính căn bản như minh họa trong hình 2-1. được gọi là "nhập, xử lý, xuất". Nghệ thuật lập trình bao gồm việc viết các lệnh để báo cho máy tính cách xử lý thông tin đặc biệt. Một vài loại dữ liệu được gửi vào chương trình, và sau đó dữ liệu được xử lý và gửi

ra khỏi chương trình. Ví dụ trong một chương trình xử lý bản lương, các giờ làm việc và lương tính giờ dành cho một số các nhân viên có thể được nhập vào. Sau đó chương trình sẽ tính tổng số chi trả gộp, và các loại khấu hao được trừ đi và chi trả vòng. Kết quả xuất sẽ tiền chi trả lương thực tế. Để quy trình xử lý được hoàn tất một cách thành công, chương trình phải thực thi các phép tính đúng theo một thứ tự đúng.



***Hình 2-1.** Dòng xử lý*

***VÍ DỤ 2.1** Tìm dữ liệu nhập, xử lý và xuất dành cho một chương trình để tính điểm trung bình của một học kỳ.*

***Nhập**: cấp về mẫu tự từ tất cả các khóa học trong học kỳ và thang chia cấp được dùng (tức là cấp A=4,0, cấp B=3,0 v.v...)*

***Xử lý**: cộng các giá trị số của tất cả các cấp thì rồi chia cho số các khóa học.*

***Xuất**: điểm trung bình dành cho kỳ thi.*

CHỦ ĐIỂM 2.2

PROBLEM SOLVING
Giải quyết vấn đề

Computer programs are written to solve problems. In English classes, writing a research paper is much easier if the outline is constructed first. The actual writing then follows the outline. In the same way, coding a computer program should not be started until all the plans are made. First, a solution is created, and then the program is coded in a computer language. The key is planning.

There are five basic steps to programming. Each step is essential in the process. Following the first two steps carefully will ultimately result in less time, energy, and frustration in the final steps.

Step 1. **Analyze the problem and develop the specifications.** The programmer must completely understand and be able to write a precise statement of the problem. In order to develop the exact specifications, there are several questions that must be asked.

- What are the inputs to the problem? What is known, what will be given, and in what form?
- What are the outputs desired? What kind of report, chart, or information is required?
- What would be the processing steps necessary to go from the given input to the desired output?

Step 2. Design a solution. A precise set of steps must be developed, that, when applied to the problem, will lead to the best solution. This is the outline from which the code will be written. The series of steps should be unambiguous, detailed and finite, able to be completed in a reasonable period of time. The solution must be complete and effective. It is helpful to test the solution by "walking through" the steps with data from Step 1 in order to be sure the solution designed is the best one possible.

Step 3. **Code the program in a programming language with documentation.** If the solution has been completely and carefully designed, translation into a programming language will be relatively straightforward. Usually, the programmer can write the code step by step exactly according to the solution. The more complete the solution, the less time and energy it takes to write the code. An integral part of the coding process, but one ignored by many programmers, is documentation. Each step of the code should be explained with comments, so the original (or a future) programmer will completely understand how the solution was reached. Both debugging during the programming process and changing the program later are

greatly facilitated by good comments embedded within the code.

Step 4. **Test the program.** This step is an iterative process with the previous step. Many novice programmers write out the entire program before testing. Then, if it doesn't work, they have no idea which section of the code has a problem. Instead, as each part of the code is written, it should be tested. For example, in the payroll program, first the programmer can write the code to accept the hours worked and hourly salary, and then test to be sure the correct values have been received by the program. It would not help to write the processing section before the inputs have been received correctly.

Step 5. **Validate the program.** Once the program has been written in its entirety, and is working for a few test cases, it should be validated by extensive testing. Just because it works for one set of test inputs does not mean it will work in every case. A broad range of test values should be applied. This is also where the user interface is examined. Are the directions sufficient? How does it handle invalid input? What happens if the user enters character data instead of numeric? Extensive fine-tuning of the program in every situation is necessary before it can be considered complete.

These five steps are essential to the problem-solving and programming process. The heart of Step 2 is the design of the algorithm.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 2.2

2.2 GIẢI BÀI TOÁN

Các chương trình máy tính được viết để giải các bài toán. Trong cách lập tiếng Anh, việc viết một trang tìm kiếm thì dễ dàng hơn nhiều. Nếu khung dàn bài đã được cấu tạo. Việc viết thực tế sau đó tuân thủ theo dàn bài này. Cũng theo cách đó việc tạo một chương trình máy tính không được bắt đầu cho đến khi tất cả vấn đề hoạch định đã được thực hiện. Trước tiên phải tạo ra một giải pháp sau đó chương trình được tạo mã theo một ngôn ngữ máy tính. Vấn đề chính yếu đó là hoạch định.

Có 5 bước căn bản để lập trình. Mỗi bước là phần thiết yếu nhất trong quy trình. Hãy tuân theo hai bước đầu tiên một cách thận trọng, để cho ta kết quả tối ưu mà rất ít tốn kém thời gian công sức và những sự nhầm lẫn ở những bước sau cùng.

Bước 1. Phân tích bài toán và phát triển các đặc trưng. Người lập trình phải hoàn toàn hiểu và có thể viết một câu lệnh chính xác về bài toán. Để phát triển các đặc trưng chính xác, có nhiều câu hỏi phải được trả lời.

1 *Dữ liệu nhập cho bài toán là gì? Những gì đã biết những gì đã cho và những gì phải tìm ?*

2 *Kết quả xuất mong muốn là gì? Loại báo cáo, biểu đồ, hoặc thông tin cần thiết?*

2 *Các bước xử lý cần thiết phải đi từ dữ liệu nhập đã cho cho đến kết quả xuất mong muốn là gì?*

Bước 2. *Thiết kế một giải pháp. Một tập hợp các bước chính xác được phá triển có nghĩa là lúc được áp dụng cho bài toán, nó dẫn đến một giải pháp tối ưu, đây chính là khung dàn bài mà từ đó mã phải được viết. Chuỗi các bước phải rõ ràng, chi tiết hóa, và hữu hạn, có thể được hoàn thành trong một khoảng thời gian hợp lý. Giải pháp phải được hoàn tất là hiệu quả, bạn phải cảm thấy hữu dụng khi thử nghiệm giải pháp bằng cách thông qua các bước dữ liệu từ bước 1 để bảo đảm được rằng giải pháp này chính là giải pháp hoàn chỉnh nhất.*

Bước 3. *Tạo mã cho chương trình theo một ngôn ngữ lập trình với tài liệu. Với giải pháp phải được hoàn chỉnh và được thiết kế cẩn thận thì việc diễn dịch thành một ngôn ngữ lập trình sẽ là một quy trình tương đối tiện lợi. Thông thường thì người lập trình có thể viết mã theo từng bước một cách chính xác phù hợp với giải pháp. Giải pháp càng hoàn chỉnh thì thời gian công sức bỏ ra để viết mã càng ít. Có một phần nguyên vẹn của quy trình tạo mã nhưng lại là một phần nhiều nhà lập trình thường bỏ qua đó là chính là tài liệu. Mỗi bước của việc tạo mã nên được giải thích bằng các lời bình chú, vì vậy người lập trình ban đầu (người lập trình trong tương lai) sẽ phải hiểu hoàn toàn cách đạt đến giải pháp này. Cả việc gỡ rối trong quy trình xử lý của chương trình và việc thay đổi chương trình sau đó đều có thể được thực hiện thông qua các lời ghi chú được trộn vào bên trong mã.*

Bước 4. *Thử nghiệm chương trình. Bước này là một quy trình lập với bước trước đó. nhiều nhà lập trình đã viết ra toàn bộ chương trình trước khi thử nghiệm. Sau đó nếu chương trình không hoạt động thì họ không có được ý tưởng mục nào trong mã gặp sự cố. Vì thế, khi mỗi phần của mã được viết, thì người ta nên thử nghiệm nó. Ví dụ trong chương trình tính lương, trước tiên nhà lập trình có thể viết mã để chấp nhận các giờ làm việc và lương tính giờ, sau đó họ phải thử nghiệm để bảo đảm rằng chương trình đã nhận được các giá trị đúng. Điều này việc viết mục xử lý trước khi nhận được các dữ liệu nhập hoàn chỉnh là điều không giúp ích được gì cả.*

Bước 5. Tạo hiệu lực chương trình. Một khi chương trình đã được viết toàn bộ, và đang hoạt động trong một vài tình huống thử nghiệm thì nó nên được tạo hiệu lực bằng phép thử nghiệm tăng cường. Điều này bởi vì nó hoạt động ứng với một tập hợp các dữ liệu vào thử nghiệm thì không có nghĩa rằng nó sẽ hoạt động trong mỗi tình huống. Nên áp dụng phạm vi các giá trị thử nghiệm rộng lớn. Đây cũng là nơi mà giao diện người dùng cần được xem xét. Các chỉ dẫn đã đầy đủ chưa? Nó có xử lý các dữ liệu nhập sai hay không? điều gì xảy ra nếu người dùng nhập vào dữ liệu ký tự thay vì dữ liệu số? Việc tăng cường canh chỉnh các chương trình trong mỗi một tình huống là cần thiết trước khi nó có thể được xem là hoàn chỉnh.

Năm bước này là năm bước thiết yếu đối với quy trình giải toán và lập trình. Trọng tâm của bước 2 là thiết kế thuật toán.

CHỦ ĐIỂM 2.3

ALGORITHMS
Các thuật toán

The goal of the problem-solving process explained above is to develop an **algorithm**, or set of steps to solve the problem. We use algorithms every day. Whenever we are doing a task, such as following a recipe or assembling a bookcase, the algorithm tells us each step in the proper order.

EXAMPLE 2.2 Write the algorithm to make a peanut butter and jelly sandwich.

1. Put the bread, peanut butter, jelly, knife, and plate onto the workspace.
2. Place two slices of bread on the plate.
3. Using the knife, spread peanut butter on one slice.
4. If you want jelly, using the knife, spread jelly on the other slice.
5. Slap the two slices together, sticky side in.
6. Repeat steps 2 through 5 for each sandwich needed.
7. Eat the sandwiches.

In the algorithm above, step 1 is the input, steps 2 through 6 explain the process, and step 7 is the output. It is clear from the algorithm that we should not do step 5 before step 3, or the sandwich would not be very interesting. Step 4 lets us make a choice whether to include jelly in the sandwich. Clearly, a number of other "if" statements could be included for choosing bananas, pickles, or any other ingredients. Also, step 6 allows us the flexibility to repeat the same process exactly several times.

To be correct, an algorithm has some specific characteristics. An algorithm:

- Has input, performs a process, and gives output.
- Must be clear and unambiguous.
- Must correctly solve the problem.
- Can be followed with paper and pencil.
- Must execute in a finite number of steps.

When writing algorithms for more complex numeric problems, usually all the input is obtained in the first few steps and the output is given in the last few steps. Many steps can be subdivided. For example, examine the algorithm in Example 2.3 below. Steps 1 through 3 are input, 4 and 5 are processing, and 6 is output. Steps 3 and 5 are subdivided into two parts for clarity.

EXAMPLE 2.3 Write the algorithm for a simple ATM machine.

1. Get the password from the user.
2. If the password is not valid, construct an error message and skip to step 6.
3. Get the inputs.
   3.1. Get the transaction type (deposit or withdrawal), and the amount from the user.
   3.2. Get the current balance from the bank.
4. If the transaction type is deposit, add the amount to the current balance.
5. If the transaction type is withdrawal, check the current balance.
   5.1. If amount is greater than the current. balance, construct an error message and skip to step 6.
   5.2. If amount is equal to or less than the current balance, subtract the amount from the current balance.
6. Output the error message or the cash, and the current balance.
7. Ask the user whether to repeat steps 3 through 6 for another transaction.

2.3.1 Pseudocode and Flowcharting

The algorithms in the previous section have been written in a structured English method called ***pseudocode***. Notice that all the lines of pseudocode contain a specific verb. In pseudocode, the steps of the algorithm are numbered in such a way that one action is executed per line. If a line requires two steps, it is subdivided in outline form to the next level (e.g., 3.1, 3..2, etc.).

Pseudocode is usually a good way to begin to design a solution to the problem. It lists the steps of the algorithm. However, it is not always easy to begin coding the program from pseudocode, because the program flow is not clear. The conditional statements and the statements to skip to another step do not provide enough of a picture of what should happen when.

In order to express the flow of processing in a much more lucid fashion, many programmers use a **flowchart**, which is a structured picture map showing the steps of the algorithm. The shapes usually used for each part of the flowchart are described in Fig. 2-2.

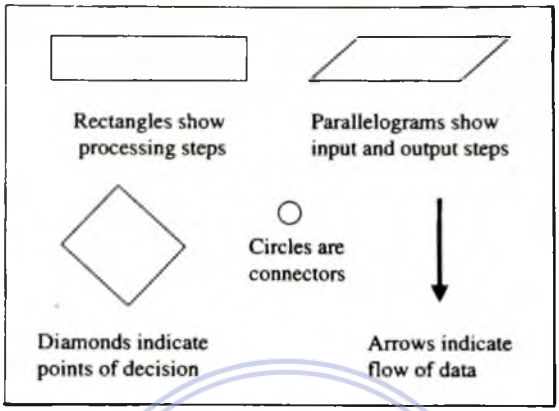
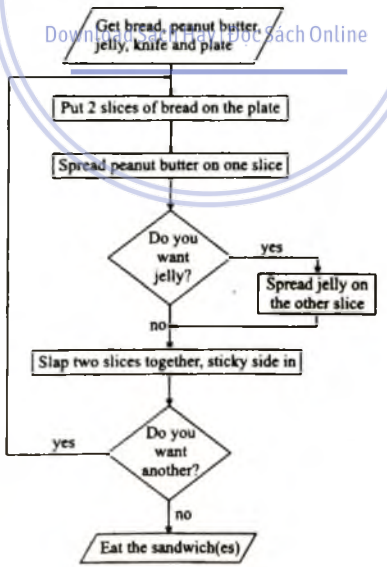


Fig. 2-2 Flowchart shapes.

EXAMPLE 2.4 A flowchart can be drawn for any algorithm. Draw a flowchart for the peanut butter sandwich algorithm shown in Example 2.2.



In the flowchart above, it is easy to identify the input and output actions because of the shapes used. The result of the decision on whether to use jelly or not is clear, either jelly is added or it is not. Also, the arrow shows the flow of control back up to the top to repeat the sandwich-making process.

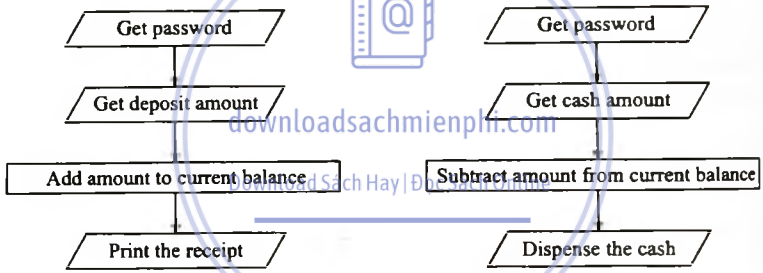
2.3.2 Basic Control Structures

Computer programs only do what the programmer has told them to do, and in that specific order. The basic control structures that are used are sequence, selection, and repetition. In Example 2.4, most steps are executed in sequence. Step 4 allows a selection and step 6 provides for repetition.

Most programs operate in sequence.

EXAMPLE 2.5 A simple flowchart for two ATM machines could be constructed as shown in Fig. 2-3. Each step, input, process, and output, is done in order, with no decisions or repetitions.

Fig. 2-3 Flowcharts for two simple ATM machines.



It is clear from the flowcharts that some decisions are necessary. The example describes two separate machines, one for withdrawal and one for deposit. Usually, you want one machine to handle both. Also, what if the password is not good? What if there is not enough money in the account for withdrawal? The flowchart needs to be constructed carefully, one section at a time.

EXAMPLE 2.6 Design one machine to determine if the transaction is a deposit or withdrawal. If the password is not good, nothing else should be done except an error message, and if there is not enough balance for the withdrawal, an error message should be given. A complete flowchart is shown in Fig. 2-4.

2.3.3 Top-down Design and Data Abstraction

The algorithms demonstrated in the previous sections are fairly simple. Most often problems cannot be solved in such a simple way. To attack more

complex problems, ***top-down design is*** used. In top-down design, the overall problem is broken into the main tasks (usually input, process, output), and then each of these tasks is broken down into subtasks until a simple solution can be seen for each subtask. Usually ***hierarchy charts*** are used as a help in top-down design. Hierarchy charts differ from flowcharts in that they indicate ***what*** should be done, not *how* the job will be accomplished. Instead of indicating the flow of data, they illustrate the tasks, or modules, for each part of the solution. The only flow of data important is what should go into and come out of each module.

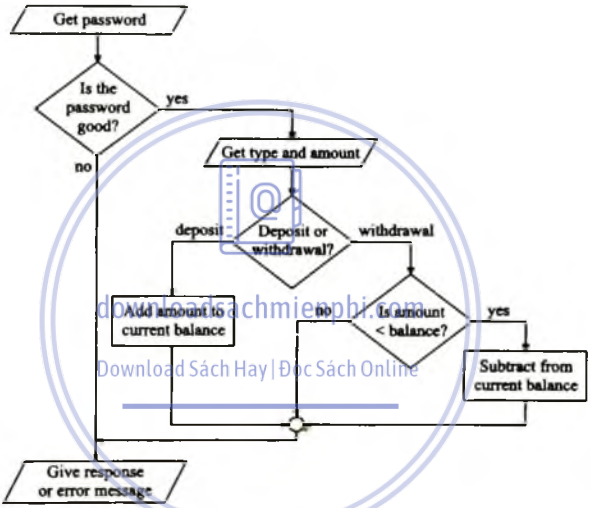


Fig. 2-4 Complete ATM machine flowchart.

EXAMPLE 2.7 Look back at the ATM program example. The overall task, or the module at the highest level, is to simulate an ATM machine. The main subtasks would be to get the inputs, to perform the calculations, and to give the outputs. The first draft of the hierarchy chart is illustrated in Fig. 2-5.

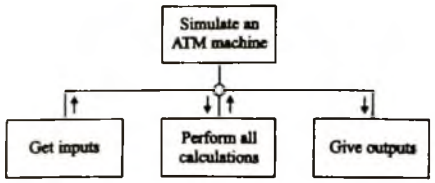


Fig. 2-5 First-level ATM hierarchy chart.

Each box represents a module of the solution. Notice, like pseudocode and flowcharts, each task begins with a verb to show the action that must be performed by that module. Arrows coming from a box indicate data coming out from that module, and arrows going into the box denote data needed by that module.

Now that the overall task and the main subtasks have been identified, each module can be subdivided even further into all the tasks that must be performed in that section. The next level for the inputs and outputs is fairly straightforward, as shown in Fig. 2-6.

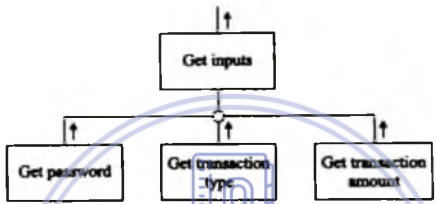


Fig. 2-6a Second-level hierarchy chart for ATM input.

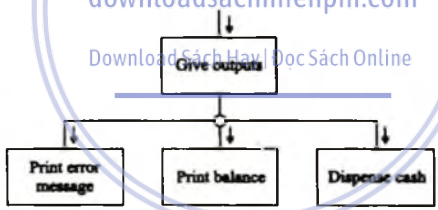


Fig. 2-6b Second-level hierarchy chart for ATM output.

The calculation module is a little more complex. There are two main subtasks, one to handle deposits and one to handle withdrawals. Each of these subtasks can be further broken down into other simpler tasks. Look at Fig. 2-7 to see the hierarchy chart for this task.

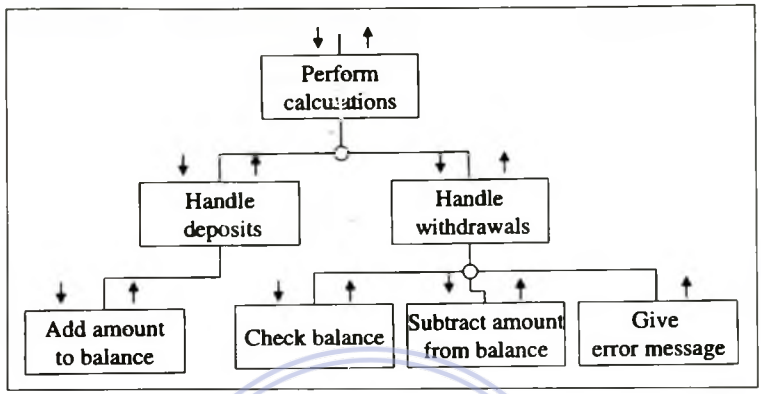
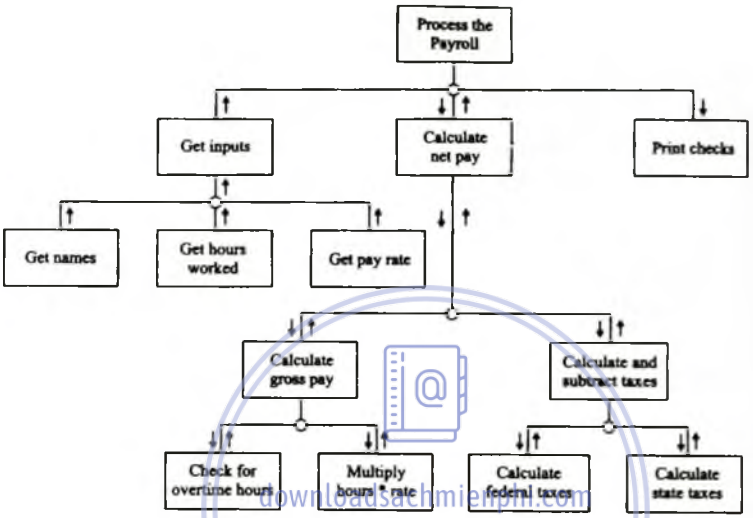


Fig. 2-7 Second- and third-level hierarchy chart for ATM calculations.

The Handle withdrawals module must check the balance to see if there is enough money in the account before adjusting the balance. If there is not enough money, an error message is generated. An error message is also needed in the module that gets the password. A complete hierarchy chart is found in Fig. 2-8.

Fig. 2-8 Complete hierarchy chart for ATM machine.

EXAMPLE 2.8 Draw the hierarchy chart for a payroll-processing program.



In the payroll chart above, the only output is to print the checks. In reality, other outputs might include the reports for federal and state taxes. Other deductions might include hospitalization or union dues. These modules would be easy to add in the appropriate place. Because it is so flexible, the hierarchy chart is uniquely suited to planning the modules for a program.

23.4 Summary of Design Tools

To review the programming process, the first thing a programmer should do is to analyze the problem and develop the specifications. Once the available inputs, required outputs, and necessary processing steps are determined, then the programmer can begin the most important part of the programming process, the planning. Tools such as pseudocode, flowcharts, and hierarchy charts aid in the process of identifying exactly what steps must be taken to solve the problem, and in what order. Only after careful planning should the actual writing of code be attempted.

The order in which these planning tools are created is often debated. Some programmers prefer writing the pseudocode first, to list in English exactly what must be done. Other programmers start with the hierarchy chart to identify the necessary modules, and then turn to pseudocode to break down each module. Usually the flowcharts are constructed last, with

a flowchart for each module. The order is not as important as using the tools to develop a complete design of the solution for the program

EXAMPLE 2.9 Go through the entire planning process to write a program that will calculate and print the average grade on three tests for an entire class.

Step 1. Analyze the problem and develop the specifications.

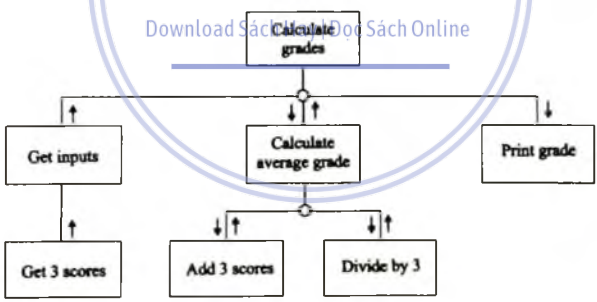
Input: 3 test scores for each student

Output: The student's average grade

Process: Add the 3 scores together and divide by 3

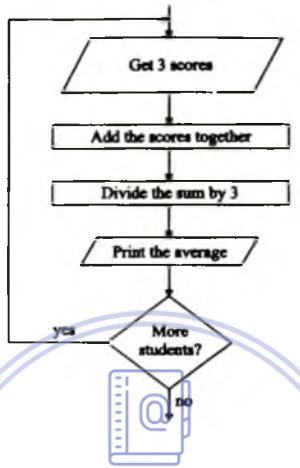
Step 2. Design a solution

1. Pseudocode:
   1. Get the 3 test scores
   2. Add the scores together
   3. Divide the sum by 3
   4. Print the average
   5. Repeat steps 1 through 4 for each student
2. Hierarchy chart:



Notice that the hierarchy chart gives no indication of looping, or how many students might be in the class. hierarchy chart only describes the modules to be written, not how many times they will be executed. The itera ' is shown in the pseudocode and flowchart.

3. **Flowchart:**



CHÚ THÍCH TỪ VỰNG

algorithm: *thuật toán*

pseudocode: *mã giả*

flowchart: *lưu đồ*

top-down design: *kiểu thiết kế từ cao xuống thấp*

hierarchy charts: *các biểu đồ phân cấp (phả hệ)*

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 2.3

2.3 THUẬT TOÁN

Mục tiêu của quy trình giải toán được giải thích trên đây đó là phải phát triển một thuật toán, hoặc một tập hợp các bước để giải quyết bài tập. Chúng ta sử dụng thuật toán hàng ngày. Bất cứ lúc nào thực hiện một tác vụ chẳng hạn như nhận hoặc lắp ráp một kệ sách, kỹ thuật toán báo cho chúng ta mỗi bước trong một trình tự hoàn chỉnh.

VÍ DỤ 2.2 *Hãy viết một thuật toán để tạo nên một bánh sandwich có bơ đậu và nhân kem.*

1. *Hãy đặt bánh mì, bơ đậu, kem, dao và đĩa vào chỗ làm việc.*

2. *Hãy đặt hai lát bánh mì trên đĩa.*
3. *Sử dụng dao rãi bơ đậu lên trên một đĩa.*
4. *Nếu muốn trang trí kem hãy sử dụng dao trãi kem trên một lát bánh mì kia.*
5. *Đặt hai lát bánh mì vào nhau.*
6. *Lặp lại các bước từ 2 đến 5 trong mỗi một miếng sandwich nếu cần.*
7. *Ăn sandwich.*

Trong thuật toán trên đây, bước 1 là nhập, bước 2 cho đến bước 6 giải thích quy trình bước 7 là xuất. Rõ ràng từ thuật toán này cho thấy không nên thực hiện bước 5 trước bước 3 hoặc bánh sandwich, nếu không thì bánh sandwich sẽ không có ngon lành. Bước 4 cho phép chúng ta chọn lựa để đưa có nên đưa kem vào bánh sandwich hay không. Rõ ràng một số các câu lệnh ít có thể được đưa vào để chọn chuối hoặc bất cứ đồ gia vị nào khác. Cũng vậy bước 6 cho phép chúng ta linh động để lặp lại quy trình giống hệt như thế nhiều lần.

Để chính xác một thuật toán phải có một vài đặc trưng chuyên biệt. Một thuật toán:

- *Phải có quy trình nhập, thực hiện một phép xử lý và xuất kết quả.*
- *Phải rõ ràng và dễ hiểu.*
- *Phải giải quyết bài toán một cách đúng đắn.*
- *Có thể ghi lại trên giấy bằng bút chì.*
- *Phải thực thi theo một trình tự các bước giới hạn*

Lúc viết thuật toán dành cho các bài tập số phức tạp, thường tất cả các dữ liệu nhập phải được tìm thấy ở một vài bước đầu tiên, và dữ liệu xuất phải được cho ở các bước sau cùng. Người ta có thể chia nhỏ nhiều bước. Ví dụ xem trước thuật toán trong ví dụ 2.3 dưới đây, bước 1 đến bước 3 là nhập, bước 4 và bước 5 là xử lý và bước 6 là xuất. Các bước 3 và 5 được chia nhỏ thành hai phần cho rõ ràng.

***VÍ DỤ 2.3** Hãy viết một thuật toán dùng cho một máy ATM đơn giản.*

1. *Nhận password từ người dùng.*
2. *Nếu password không đúng, thì tạo nên một thông điệp báo lỗi và lướt sang bước 6.*
3. *Nhận dữ liệu nhập*
   1.1 *Nhận kiểu chuyển khoản (gửi hay rút) và lượng tiền do người dùng nhập vào.*

1.2 Nhận bảng cân đối tiền hiện tại ở ngân hàng.

4. *Nếu kiểu giao dịch là tiền gửi thì cộng thêm lượng tiền vào bảng cân đối tài khoản hiện tại.*
5. *Nếu kiểu giao dịch là rút tiền thì hãy kiểm tra bảng cân đối tài khoản hiện tại.*

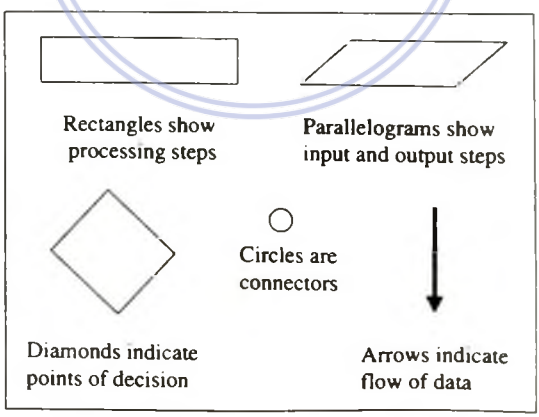
1.3 Nếu lượng tiền rút ra lớn hơn lượng tiền đang có, thì thuật toán cấu tạo nên một thông điệp báo lỗi và bước sang bước 6.

1.4 Nếu lượng tiền này bằng hoặc nhỏ hơn lượng tiền đang có trong bảng cân đối, thì trừ lượng rút khỏi lượng tiền hiện tại.

6. *Xuất thông điệp báo lỗi hoặc xuất tiền mặt và cân đối lại tiền hiện tại.*
7. *Yêu cầu người dùng phải lặp lại các bước từ bước 3 cho đến bước 6 cho một khoản giao dịch rút tiền khác.*

2.3.1 Tạo mã giả và lưu đồ

Các thuật toán dùng trong mục trước đây được viết theo một phương pháp cấu trúc tiếng Anh được gọi là mã giả (pseudocode). Lưu ý rằng tất cả các dòng mã giả đều có chứa một động từ đặc biệt. Trong mã giả các bước của thuật toán được đánh số theo một cách thức qua đó một hành động được thực thi trên mỗi dòng. Nếu một dòng yêu cầu phải có hai bước, thì nó được chia nhỏ thành các dạng khung dàn bài sang bước kế tiếp. (ví dụ 3.1, 3.2 v.v...)



***Hình 2-2** Hình dạng của lưu đồ*

Mã giả thường là cách tốt để bắt đầu thiết kế một giải pháp cho bài toán. Nó liệt kê các bước của thuật toán. Tuy nhiên, chúng ta không

dễ dàng để viết mã chương trình từ mã giả. Bởi vì dòng chương trình không rõ ràng, các câu lệnh điều khiển và các câu lệnh để lướt sang bước khác không có cung cấp đủ hình ảnh cho những gì sẽ xảy ra. Để trình bày dòng xử lý theo một cách thức dễ hiểu hơn, nhiều nhà lập trình đã sử dụng một lưu dồ đây là một bản đồ hình ảnh có cấu trúc biểu thị các bước về thuật toán. Các hình dáng được dùng cho mỗi phần của lưu đồ được mô tả trong hình 2-2.

***VÍ DỤ 2.4** Một lưu đồ được vẽ cho bất kỳ một thuật toán nào. Hãy vẽ một lưu đồ dành cho thuật toán tạo bánh sandwich có nhân bơ như minh họa trong ví dụ 2.2.*

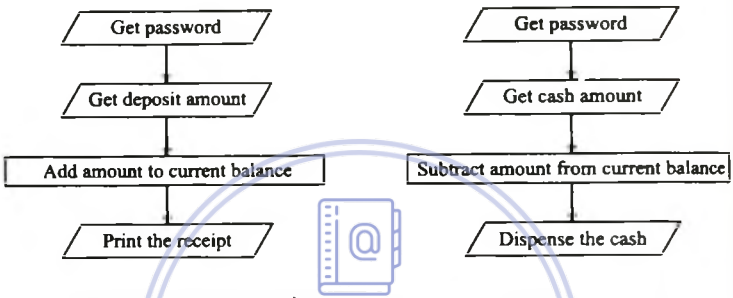
Trong lưu đồ trên đây ta dễ dàng nhận biết các hành động nhập và xuất do bởi hình dáng được dùng. Kết quả của quyết định có nên sử dụng kem hay là không, thật rõ ràng, mỗi một kem trang trí được thêm vào hay không được thêm vào. Cũng vậy mũi tên chỉ cho biết dòng điều khiển lên đến đỉnh để lặp lại quy trình làm bánh sandwich.

2.3.2 Các cấu trúc điều khiển căn bản

Các chương trình máy tính chỉ thực hiện những gì mà những nhà lập trình bảo chúng ta phải làm và phải tuân theo một trình tự đặc biệt. Các cấu trúc điều khiển căn bản được dùng là thứ tự, chọn lựa và lặp

lại. Trong ví dụ 2.4 hầu hết các bước được thực thi trong một trình tự. Bước 4 cho phép chọn lựa và bước 6 cung cấp xác lập. Tất cả các chương trình đều thực hiện theo một chuỗi trình tự.

***VÍ DỤ 2.5** Một lưu đồ đơn giản đành cho hai máy ATM có thể được cấu tạo như minh họa trong hình 2.3. mỗi bước nhập, xử lý và xuất được thực hiện theo thứ tự mà không có quyết định hoặc sự lặp lại nào.*



***Hình 2-3** Các lưu đồ dành cho hai máy ATM đơn giản*

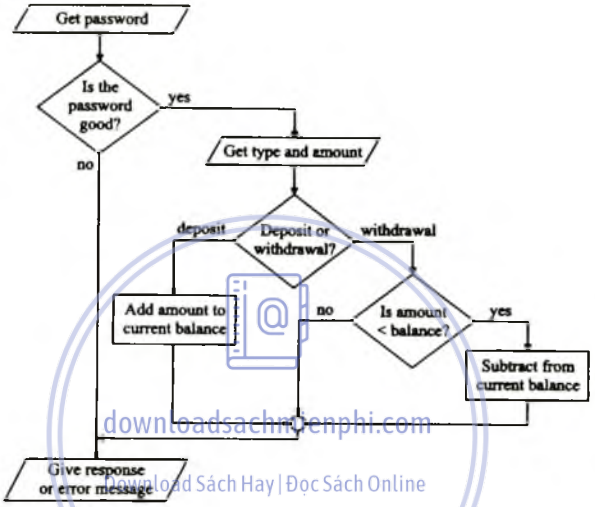
Từ lưu đồ cho thấy rõ ràng rằng một vài quyết định là cần thiết. Ví dụ này mô tả hai máy chuyên biệt, một máy để rút tiền và một máy để gửi tiền. Thường bạn muốn một máy xử lý cả hai chức năng. Cũng vậy điều gì xảy ra nếu password không tốt? Điều gì xảy ra nếu không có đủ lượng tiền dành cho người rút tiền? Lưu đồ cần phải được cấu tạo một cách cẩn thận, một chuyên mục một lần.

***VÍ DỤ 2.6** Hãy thiết kế một máy để xác định xem thử lượng tiền là được gửi vào hay được rút ra. Nếu password không đúng, thì không có gì được thực hiện ngoại trừ thông điệp lỗi, và nếu không có đủ lượng tiền mặt để rút ra cũng có một thông điệp báo lỗi cung cấp. Lưu đồ hoàn chỉnh được minh họa trong hình 2-4.*

2.3.3 Thiết kế top – down và sự tính trừu tượng của dữ liệu

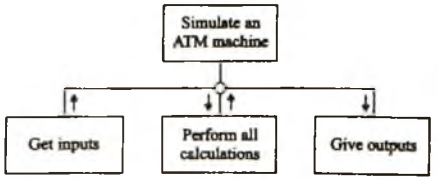
Thuật toán được minh họa trong phần trên đây rõ ràng là đơn giản. Hầu hết các bài toán thường gặp không được giải theo một cách thức đơn giản như thế. Để thâm nhập vào những bài toán phức tạp hơn, người ta sử dụng thiết kế top – down. Trong thiết kế top – down, toàn bộ vấn đề được ngắt thành những tác vụ chính (thường là nhập, xử lý, xuất) sau đó mỗi một tác vụ lại được ngắt thành những tác vụ con cho đến khi tìm thấy được một giải pháp đơn giản cho mỗi một tác vụ con. Thông thường các sơ đồ phả hệ được dùng để trợ giúp cho thiết kế top – down. Các sơ đồ phả hệ khác với các lưu đồ ở chỗ là chúng chỉ định những gì nên làm, chứ không chỉ định cách hoàn thành công việc.

Thay vì chỉ ra dòng dữ liệu thì chúng lại minh họa các tác vụ hoặc các module dành cho mỗi một phần của giải pháp. Dòng dữ liệu quan trọng duy nhất là những gì nên đưa vào và những gì xuất ra mỗi module.



***Hình 2-5** Sơ đồ hoàn chỉnh của một máy ATM*

***VÍ DỤ 2.7** Hãy xem lại ví dụ chương trình ATM. Tác vụ tổng thể hoặc module ở tại mức cao nhất đó là phải mô phỏng một máy ATM. Những tác vụ con chính sẽ nhận dữ liệu nhập, thực hiện các phép tính và cho kết quả xuất. Phác thảo đầu tiên của sơ đồ phả hệ được minh họa trong hình 2-5.*

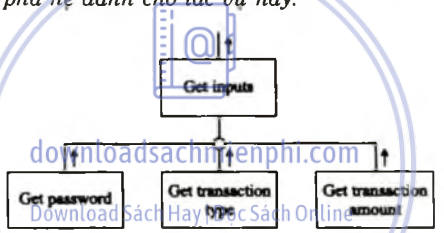


***Hình 2-5** Sơ đồ phả hệ ATM ở cấp đầu tiên*

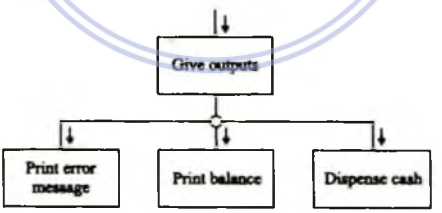
Mỗi một ô trình bày một module của giải pháp. Lưu ý cũng giống như mã giả và lưu đồ, mỗi tác vụ bắt đầu một động từ để biểu thị hành động phải được thực hiện do module đó. Các mũi tên để xuất phát từ một ô chỉ ra dữ liệu xuất ra khỏi module đó và các mũi tên đi vào một ô chỉ cho biết dữ liệu mà module đó cần thiết.

Lưu ý rằng tác vụ tổng thể và các tác vụ con chính phải được nhận biết mỗi module có thể chia nhỏ đồng đều nhau nó chuyên sâu hơn thành tất cả các tác vụ mà mục đó phải thực hiện. Mức kế tiếp dành cho dữ liệu nhập và xuất, rõ ràng là thuận lợi như minh họa trong hình 2-6.

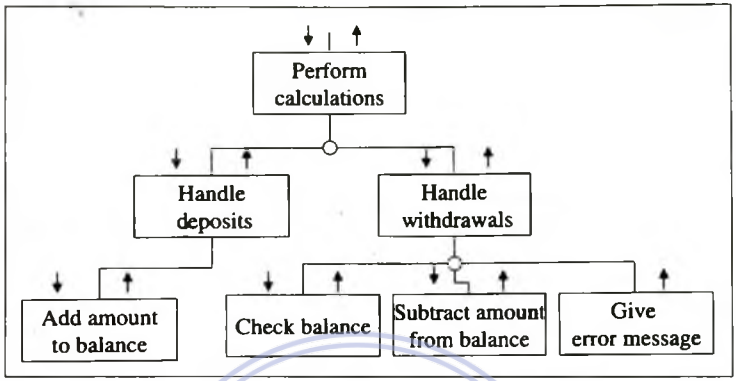
Module tính toán thì phức tạp hơn. Có hai tác vụ con, một để xử lý tiền gửi và một để xử lý tiền rút ra. Mỗi trong số tác vụ con này có thể phân chia nhỏ thành các tác vụ đơn giản hơn. Hãy xem hình 2-7 để biết được sơ đồ phả hệ dành cho tác vụ này.



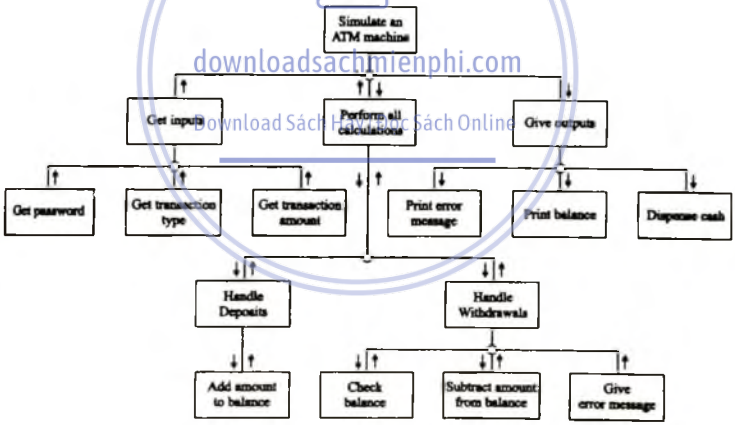
***Hình 2.6a** Sơ đồ phả hệ hai cấp dành cho các kết quả nhập ATM*



***Hình 2.6b** Sơ đồ phả hệ hai cấp dành cho các kết quả xuất ATM*



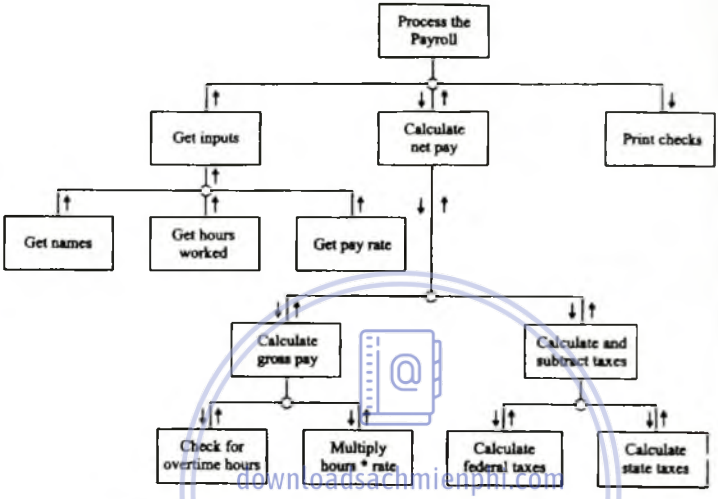
Hình 2-7 *Sơ đồ phả hệ hai cấp và ba cấp dành cho các phép tính ATM*



Hình 2.8 *Sơ đồ phả hệ hoàn chỉnh dành cho máy ATM.*

Xử lý module rút tiền mặt phải kiểm tra bảng cân đối để xem thử có đủ lượng tiền trong tài khoản không trước khi điều chỉnh bảng cân đối này. Nếu không đủ lượng tiền, thì mục thông điệp báo lỗi được tạo ra. Mục thông điệp báo lỗi cũng cần trong module này để nhận password. Một sơ đồ phả hệ hoàn chỉnh được cho trong hình 2-8.

VÍ DỤ 2.8 *Vẽ một sơ đồ phả hệ dành cho một chương trình xử lý bảng lương*



Trong sơ đồ xử lý bảng lương trên đây, chỉ có kết quả xuất là in các ngân phiếu. Trong thực tế thì các kết quả xuất có thể chứa các bản báo cáo về các khoản thuế trong nước. Những khoản khấu hao khác cũng được đưa vào chẳng hạn như khấu hao về chi phí bệnh viện hoặc khấu hao về nghiệp đoàn. Những module này dễ dàng bổ sung ở những chỗ phù hợp. Bởi vì nó quá sinh động cho nên sơ đồ phả hệ thường phù hợp để hoạch định các module dành cho một chương trình.

2.3.4 Khái quát về các công cụ thiết kế

Để xem lại phương pháp lập trình điều đầu tiên một nhà lập trình phải làm đó là phân tích vấn đề và phát triển các đặc trưng, một khi đã có sẵn dữ liệu nhập, các kết quả xuất cần thiết cũng như các bước xử lý cần thiết phải được xác định, sau đó người lập trình có thể bắt đầu phần quan trọng nhất của phương pháp lập trình đó là hoạch định. Các công cụ chẳng hạn như mã giả lưu đồ, và sơ đồ phả hệ trợ giúp trong quy trình nhận biết đích xác bước nào phải được tiến hành để giải quyết vấn đề và tiến hành theo thứ tự như thế nào. sau khi hoạch định cẩn thận bạn mới thật sự viết mã.

Thứ tự mà qua đó các công cụ hoạch định này được tạo ra thường là tùy ý, một số nhà lập trình thích viết mã giả trước, để liệt kê danh

sách theo tiếng Anh một cách chính xác những gì phải làm. Những nhà lập trình khác lại bắt đầu với sơ đồ phả hệ để nhận biết các module cần thiết rồi trả về mã giả để ngắt chia nhỏ mỗi module. Thường thì các lưu đồ được cấu tạo sau cùng, một lưu đồ dành cho mỗi module. Thứ tự không quan trọng cho bạn việc sử dụng các công cụ để phát triển một bản thiết kế hoàn chỉnh giải pháp dành cho chương trình.

VÍ DỤ 2.9 *Thông qua toàn bộ quy trình hoạch định hãy viết một chương trình để tính và in điểm trung bình trong ba kỳ thi dùng cho một lớp.*

Bước 1. Phân tích bài toán và phát triển phép đặc trưng

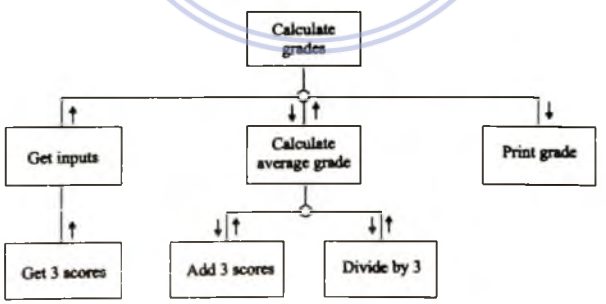
Nhập: 3 điểm thi cho mỗi một học viên

Xuất: điểm trung bình của học viên

Xử lý: cộng ba điểm thi lại với nhau và chia cho 3.

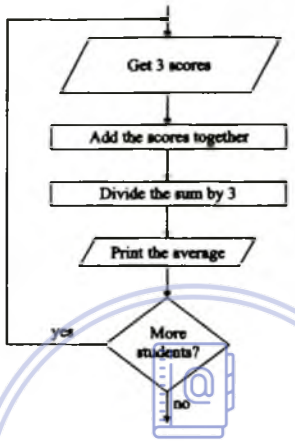
Bước 1. Thiết kế một giải pháp

1. *Mã giả*
   a. *Nhận ba điểm thi*
   b. *Cộng điểm thi lại với nhau*
   c. *Chia tổng cho 3*
   d. *Chia điểm trung bình*
   e. *Lặp lại bước từ 1 đến 4 cho mỗi một học viên.*
2. *Sơ đồ phả hệ*



Lưu ý rằng sơ đồ phả hệ không cho chúng ta đặc trưng của vòng lặp hoặc số lần mà học sinh có mặt trong lớp. Sơ đồ phả hệ chỉ mô tả module phải được viết chứ không trình bày số lần mà chúng thực thi. Vòng lặp được minh họa trong mã giả và trong lưu đồ.

3. *Lưu đồ*



SOLVED PROBLEMS
Bài tập có lời giải

2.1 What would be the input, process, and output for a program to change Fahrenheit temperatures into Celsius?

Input: Temperature in Fahrenheit

Output: Temperature in Celsius

Process: Apply the formula Celsius = 5/9 * (Fahrenheit - 32)

2.2 True or False. The first thing a good programmer does when asked to write a computer program is to sit down at the computer and begin typing in code.

False. Coding does not begin until the problem is examined, analyzed, and a complete solution is designed. .

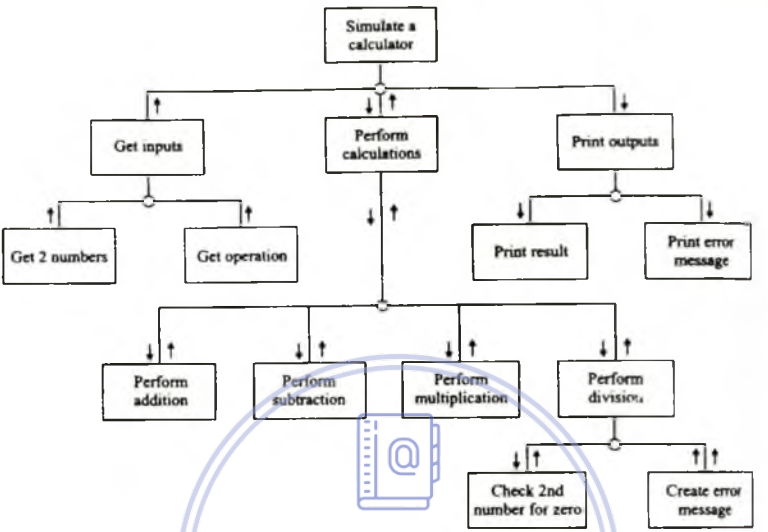
2.3 Write the algorithm in pseudocode for a simple calculator.

1. Get two numbers and the operation desired
2. Check the operation
   - 2.1. If the operation is addition, the result is the first + the second
   - 2.2. If the operation is subtraction, the result is the first - the second
   - 2.3. If the operation is multiplication, the result is the first * the second
   - 2.4. If the operation is division, check the second number

     If the second number is zero, construct an error message

     If the second number is not zero, the result is the first / the second
3. Print out the result or the error message

2.4 Draw the hierarchy chart for a simple calculator.



2.5 Go through the entire planning process to write a program that will handle the checkout for a hotel room. This hotel charges $55 for one person, $60 for two, and $65 for three or more.

Step 1. **Analyze the problem and develop the specifications.**

Input: Number of nights, number of people in the room, meal charges

Output: The printed bill, including tax

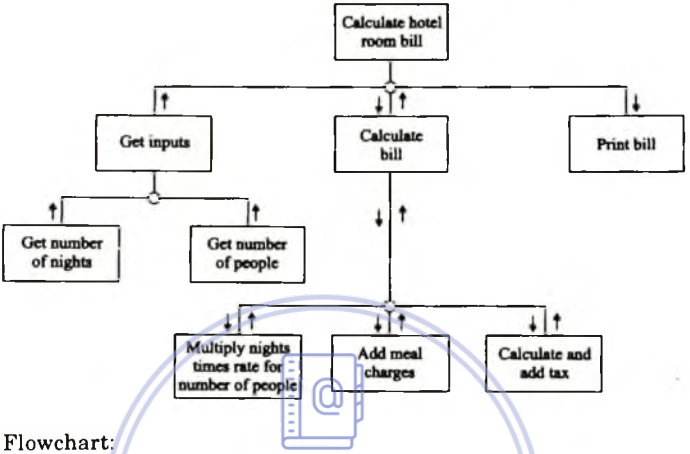
Process: Calculate the room charges, add the meal charges, and then add the tax

Step 2. **Design a solution.**

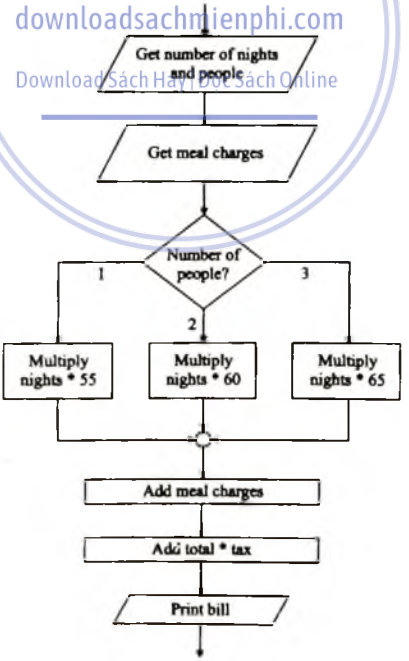
Pseudocode:

1. Get the number of nights and number of people in the room from the customer
2. Get the meal charges from the restaurant
3. Calculate bill
   - 3.1. Look up the room charge for that number of people
   - 3.2. Multiply by the number of nights
   - 3.3. Add the meal charges
   - 3.4. Calculate and add the tax
4. Print the bill

Hierarchy chart:

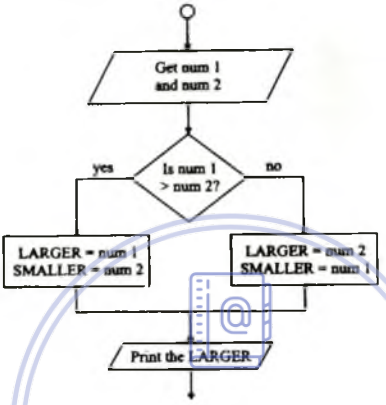


Flowchart:



2.6 Draw a flowchart from the following pseudocode to input two numbers and print the larger.

1. Get numberl and number2
2. Compare to see which is larger
3. Print the larger number

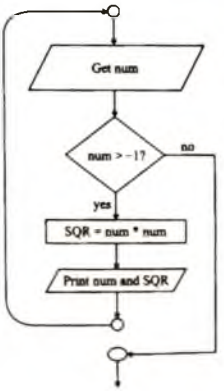


2.7 Write an algorithm in pseudocode and the flowchart to input a number and print the number and its square until the user enters a negative number.

Pseudocode:

1. Get the number
2. Calculate its square
3. Print the number and its square
4. Repeat steps 1 through 3 until the person enters -1

Flowchart:

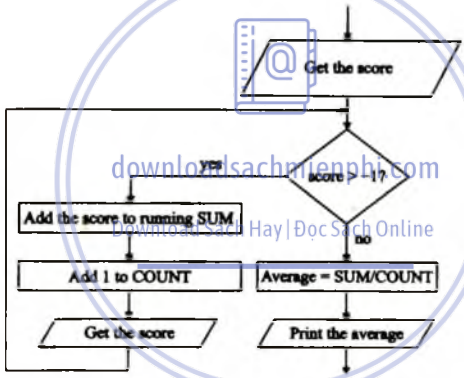


2.8 Write an algorithm in pseudocode and the flowchart to input a series of test scores until the user enters negative one (-1) and then print out the average score of all the numbers. You do not know how many numbers are coming into the program.

Pseudocode:

1. Get the score
2. Add it to the running sum
3. Add one to the count of scores
4. Repeat steps 1 through 3 until the person enters -1
5. Calculate average by dividing running sum by the count of scores
6. Print out the average

Flowchart:



HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

2.1 Tìm dữ liệu nhập, xử lý và xuất dành cho một chương trình để thay đổi nhiệt độ Fahrenheit sang độ Celsius?

Nhập: nhiệt độ theo Fahrenheit

Xuất: nhiệt độ theo Celsius

Xử lý: áp dụng công thức Celsius bằng 5/9 (Fahrenheit -32)*

2.2 Có phải điều sau đây đúng hay sai. Điều đầu tiên mà một người lập trình tốt cần phải làm lúc được yêu cầu viết một chương trình máy tính đó là phải ngồi tại máy tính và bắt đầu gõ nhập mã.

2.3 Hãy viết thuật toán theo mã giả dành cho một máy tính đơn giản

2.4 *Vẽ sơ đồ phả hệ dành cho một máy tính đơn giản.*

2.5 *Thông qua toàn bộ quy trình hoạch định để viết một chương trình xử lý tiền chi trả cho một phòng thuê. Chi phí phòng ở khách sạn là 55 đô cho một người, 60 đô cho hai người và 65 đô cho 3 người hoặc nhiều hơn.*

Bước 1. Phân tích bài toán và phát triển các đặc trưng

Nhập: số đêm, số người trong phòng, chi phí bữa ăn.

Xuất: hóa đơn được in ra kể cả thuế

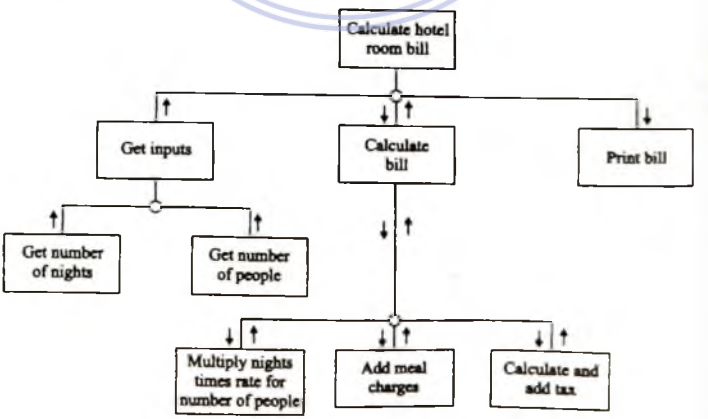
Xử lý: tính toán số tiền mướn phòng cộng với chi phí bữa ăn và tiền thuế.

Bước 2. Thiết kế một giải pháp

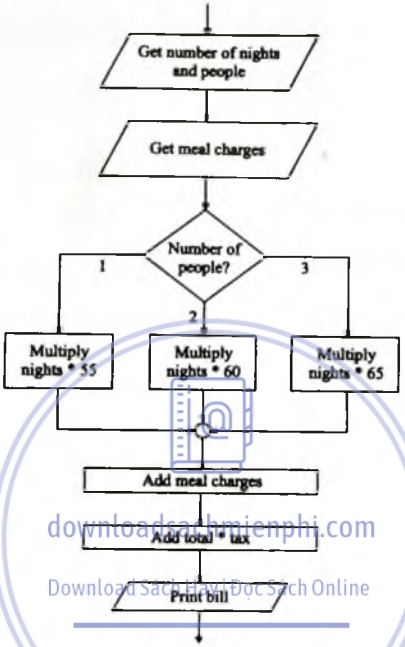
Mã giả:

1. *Nhận số đêm và số người trong phòng từ khách hàng.*
2. *Nhận chi phí bữa ăn từ nhà ăn.*
3. *Tính toán hóa đơn*
   - *1.1 Xem chi phí phòng để biết số người.*
   - *1.2 Nhân cho số đêm ở*
   - *1.3 Cộng chi phí bữa ăn*
   - *1.4 Tính và cộng thuế*
4. *In hóa đơn*

Lưu đồ phả hệ



Lưu đồ:



2.6 *Vẽ một lưu đồ từ mã giả sau đây để đưa vào hai số và in số lớn hơn.*

2.7 *Viết một thuật toán theo mã giả và lưu đồ để nhập một số và in một con số và bình phương của nó đến khi người dùng gặp một số âm.*

2.8 *Viết một thuật toán theo mã giả và lưu đồ để nhập vào một chuỗi các điểm thi cho đến khi người dùng gặp số (–1) rồi in điểm trung bình của tất cả các số. Bạn không biết được có bao nhiêu số đưa vào chương trình.*

Mã giả.

1. Ghi điểm

2. Thêm nó vào tổng đang tính.

3. Tăng biến đếm điểm thêm 1

4. Lặp lại từ bước 1 đến bước 3 cho đến khi nào tên gõ nhập là -1

5. Tính giá trị trung bình bằng cách lấy tổng chia cho số đếm

6. In giá trị trung bình này ra.

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

2.9 What would be the input, process, and output for a program to calculate the miles per gallon for a specific vehicle?

2.10 What is the last step of the programming process? Why is it important?

2.11 Write the algorithm in pseudocode for a program to calculate the area of simple shapes.

2.12 Draw the hierarchy chart for a program to calculate the area of simple shapes.

2.13 The Bigg family is having a reunion. They want to know how many people will attend the reunion, and also the percentage of attendance broken into these age groups: preschool age (0 through 5), school age (6 through 12), teens (13 through 19), adults (20 through 64) and seniors (65 and up). Go through the entire planning process to write a program that will calculate the attendance by age groups.

2.14 Draw a flowchart from the following pseudocode to input three numbers and print out their sum and their average.

1. Get number1, number2, and number3.
2. Add them together for the sum.
3. Divide the sum by 3 for the average.
4. Print the sum and the average.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

2.9 *Tìm dữ liệu nhập, xử lý, và xuất dành cho một chương trình để tính số dòng trên một gallon ứng với một chiếc xe đặc biệt.*

2.10 *Bước cuối cùng trong quy trình xử lý chương trình là gì? Tại sao nó quan trọng?*

2.11 *Viết thuật toán theo mã giả dành cho một chương trình để tính diện tích của một hình đơn giản.*

2.12 *Hãy vẽ một lưu đồ dành cho một chương trình để tính diện tích của hình đơn giản.*

2.13 Gia đình Bigg đang có một buổi họp mặt đoàn tụ. Họ muốn biết có bao nhiêu người phải tham gia buổi họp mặt đoàn tụ này và tỉ lệ phần trăm số người tham dự được phân chia thành các nhóm tuổi: tuổi trước khi đến trường (từ 0 - 5). Tuổi đang đi học (từ 6 - 12), tuổi thiếu niên (13 - 19), tuổi trưởng thành (20-64) và người lớn tuổi (trên 65). Thông qua quy trình hoạch định toàn bộ hãy viết một chương trình để tính số người tham gia theo từng nhóm tuổi.

2.14 Hãy vẽ một lưu đồ từ mã giả sau đây để nhập vào ba số và in tổng của chúng. Cũng như số trung bình

a. Nhận number1, number2 và number3

b. Cộng chúng lại với nhau để tìm tổng

c. Chia tổng cho 3 để tìm giá trị trung bình

d. In tổng và giá trị trung bình.

ANSWERS TO SUPPLEMENTARY PROBLEMS
Trả lời các bài tập bổ sung

2.9 Input: The distance traveled, and the number of gallons used by a specific vehicle

Process: Divide the distance by the number of gallons

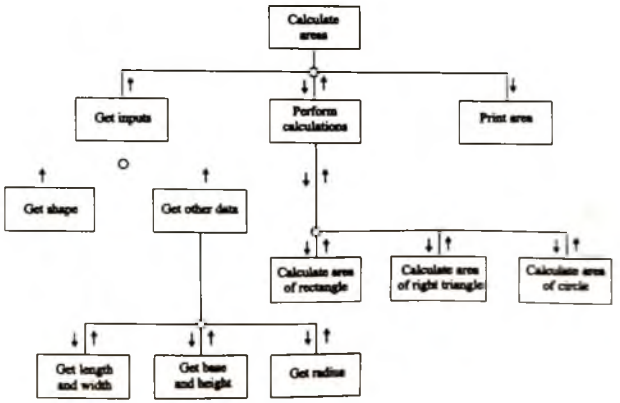
Output: The miles per gallon for that vehicle

2.10 The last step is the validation of the program. Just because the program works with one set of data does not mean it performs correctly in every situation. The programmer cannot be confident that the program works well until a broad range of test cases have been examined.

2.11 The algorithm would be:

1. Get the shape desired
2. If the shape is a rectangle
   - 2.1. Get the length and width
   - 2.2. Calculate the area as length * width
3. If the shape is a right triangle
   - 3.1. Get the base and height
   - 3.2. Calculate the area as base * height / 2
4. If the shape is a circle
   - 4.1. Get the radius
   - 4.2. Calculate the area as radius * radius * 3.14
5. Print out the area

2.12 The hierarchy chart:



2.13 The program for the Bigg family:

Step 1. **Analyze the problem and develop the specifications.**

Input: Ages of each person attending

Output: The chart of people in each age group

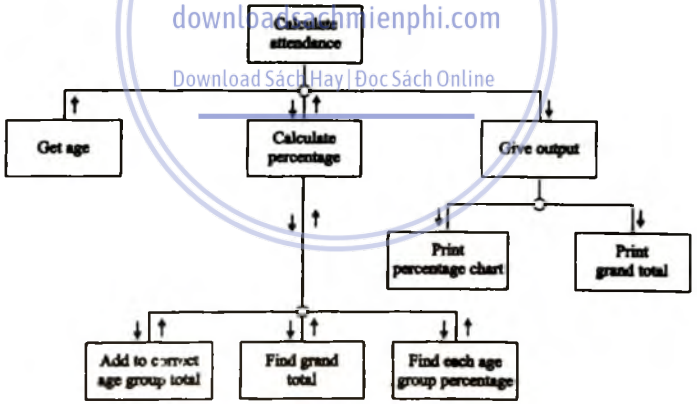
Process: Add 1 to the age group for each person's age

Step 2. **Design a solution.**

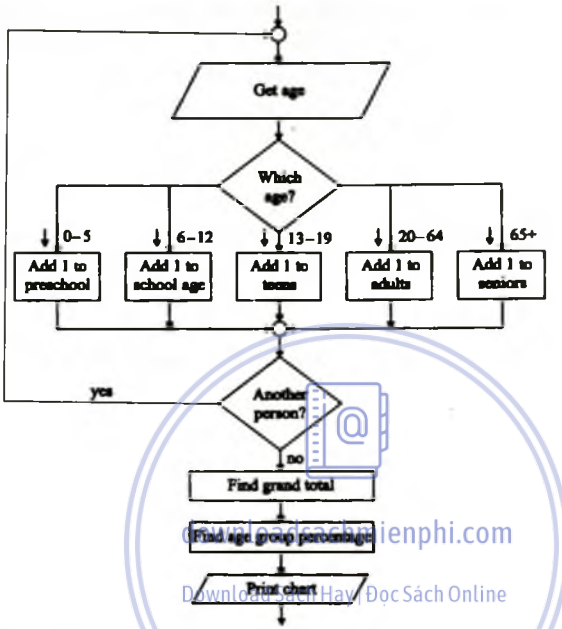
Pseudocode:

1. Get the age of the person
2. Add it to the appropriate age group
3. Repeat steps 1 and 2 for each person coming
4. Calculate percentage for each age group
   - 4.1. Find the grand total for all age groups
   - 4.2. Divide each age group total by the grand total
5. Print the chart of attendance

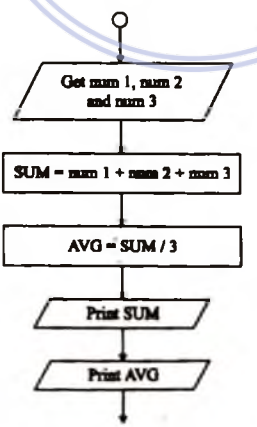
Hierarchy chart:



Flowchart



2.14 Flowchart.



CHAPTER 3

Program Coding and Simple Input/Output

Viết mã chương trình và các lệnh nhập/xuất đơn giản

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các vấn đề liên quan đến việc viết mã cho chương trình, các lệnh nhập và xuất đơn giản, cụ thể là các nội dung cơ bản sau đây:

- Programming languages
- Variables and constants
- Assignment statements
- Aritthhmetic expressions operator precedence
- Comment statements
- Simple input/output
- Writing a complete program

- *Các ngôn ngữ lập trình*
- *Các biến và hằng*
- *Các lệnh gán*
- *Các biểu thức số học và thứ tự ưu tiên của toán tử*
- *Các câu lệnh chú thích*
- *Nhập/xuất đơn giản*
- *Viết một chương trình hoàn chỉnh*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.

CHỦ ĐIỂM 3.1

PROGRAMMING LANGU
Các ngôn ngữ lập trìn

Recall the steps of the programming process listed

- Step 1. Analyze the problem and develop the
- Step 2. Design a solution.
- Step 3. Code the program in a programming mentation.
- Step 4. Test the program.
- Step 5. Validate the program.

Once the problem is thoroughly analyzed and th pletely designed, then the third step, that of writing is started. The algorithms and flowcharts help us l stand how to solve the problem. But the computer n gram written in machine language that it can unders do not have to write all our programs in the machine's source code that a translating program changes into

There are two kinds of translating programs, com ers. ***Compilers*** take the entire source code progra machine language. Only after the source code is tra can the computer execute the program. ***Interpreters*** each line, one at a time. If an error is encountered, that point. These translators, both compilers and i literal and specific. A more general explanation of th is found in Appendix A.

Our source code must be exact to be understood order to make it equally clear for human beings, con into the code that the computer ignores. This ***docum*** be explained later in the chapter.

The languages addressed in this and the followin Basic (Microsoft's structured dialect of the older Bas and Java. These languages are very different. C an usually compiled and Visual Basic is interpreted. In b programs (executables) can be created to run on c have a C/C++ compiler or a Visual Basic interprete ability of the program is usually limited to the same c in the original translating process.

On the other hand, Java is a language that is both compiled and interpreted. The source code is compiled, not into machine language, but into a bytecode that can be interpreted by many different kinds of operating systems. The interpreter on any other computer. Tzhen translates the bytecode into the machine language needed by that system. Therefore, Java has the advantage of being more portable than other languages. Certain kinds of Java programs, called **applets,** can even be embedded into pages sent over the World Wide Web and run by any Java-aware Internet browser.

All three of these languages use the same alphabet which includes any character found on the computer keyboard. Any of these characters must be translated into machine language. For instance, when you type the letter "t", or press the spacebar, your message must be represented in the computer's memory in some way. A standard code has been developed with a specific numeric representation of every character that can be used by programming languages. The American Standard Code for Information Interchange (ASCII) is consistently used by most computers and peripheral devices. The numbers from 0 to 255 represent each letter, number, space, punctuation mark, and other character available to the programming languages. The portion of the ASCII chart usually used is seen in Table 3-1. ASCII 0 through 31, and also 12,7, are unprintable characters representing various messages that can be sent. For example, 12 sends a form feed to the printer and 7 makes a beeping sound.

Table 3-1 ASCII Character Set.

Decimal	Character	Decimal	Character	Decimal	Character	Decimal	Character
0	nul	32	space	64	@	96	`
1	soh	33	!	65	A	97	a
2	stx	34	"	66	B	98	b
3	etx	35	#	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g
8	bs	40	(	72	H	104	h
9	ht	41	)	73	I	105	i
10	nl	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	soh	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	p
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w

24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.1

3.1 CÁC NGÔN NGỮ LẬP TRÌNH

Hãy nhớ lại các bước trong phương pháp lập trình được liệt kê trong chương 2.

- *Bước 1. Phân tích bài toán và phát triển các đặc trưng.*
- *Bước 2. Thiết kế một giải pháp*
- *Bước 3. Viết mã cho chương trình theo một ngôn ngữ lập trình với tài liệu*
- *Bước 4. Thử nghiệm chương trình*
- *Bước 5. Tạo hiệu lực đúng cho chương trình*

Mỗi khi bài toán đã được phân tích thuật toán được thiết kế hoàn toàn thì bước 3 tức là bước viết chương trình theo mã phải được bắt đầu. Các thuật toán và các lưu đồ giúp cho chúng ta là những con người hiểu được cách giải quyết vấn đề. nhưng máy tính cần phải có một chương trình được viết theo ngôn ngữ máy để có thể hiểu được. May thay chúng ta không phải viết tất cả các chương trình của mình theo các chữ số 1 và số 0 của máy. Chúng ta viết mã nguồn để một chương trình diễn dịch sẽ thay đổi thành ngôn ngữ của máy.

Có hai loại chương trình diễn dịch đó là trình biên soạn và trình biên dịch. Trình biên soạn nhận toàn bộ trình mã nguồn và thay nó thành ngôn ngữ máy. Chỉ sau khi mã nguồn đã được dịch toàn bộ thì máy tính mới có thể thực thi chương trình. Các bộ phiên dịch sẽ dịch và thực thi mỗi dòng, một dòng một lần. Nếu gặp phải lỗi chương trình sẽ ngưng tại điểm đó. Những bộ diễn dịch cả bộ biên soạn lẫn bộ biên dịch đều rất từ chương và chuyên biệt. Phần giải thích tổng quát hơn về quy trình dịch sẽ được cho trong phụ lục A.

Mã nguồn của chúng ta phải được máy tính hiểu một cách chính xác. Để làm cho nó rõ ràng, các lời bình chú được trộn vào trong mã để máy tính bỏ qua. Tài liệu về mã sẽ được giải thích ở phần sau trong chương này.

Ngôn ngữ được đề cập trong chương này và trong những chương sau là Visual Basic (thuật ngữ microsoft để chỉ ngôn ngữ Basic trước dây), C, C++ và Java. Những ngôn ngữ này rất khác nhau. Các mã nguồn của C và C++ thường được biên soạn và Visual Basic sẽ được diễn dịch. Trong cả hai trường hợp chương trình đứng riêng (có thể thực thi) có thể được tạo để chạy trên các máy tính không có một trình biên soạn C, C++ hoặc không có một trình biên dịch Visual Basic. Tuy nhiên, tính khả thi của chương trình là bị giới hạn sang hệ điều hành được dùng trong hệ điều hành giống hệt như hệ điều hành được dùng trong quy trình diễn dịch gốc.

Mặc khác, Java là một ngôn ngữ được biên soạn và được biên dịch. Mã nguồn được biên soạn, không phải biên soạn thành một ngôn ngữ máy nhưng biên soạn thành một mã bài có thể được diễn dịch bởi nhiều loại hệ điều hành khác nhau. Bộ biên dịch trên bất kỳ máy nào khác sẽ diễn dịch mã byte thành ngôn ngữ máy mà hệ thống đó cần thiết. Do đó Java có ưu điểm là linh động hơn các ngôn ngữ khác. Một vài loại chương trình Java có tên là Applets thậm chí có thể được nhúng vào các trang được gửi trên World Wide Web sau đó được bất cứ bộ trình duyệt Internet nào chạy tất cả ba trong số ngôn ngữ này sử dụng bảng mẫu tự chữ cái giống nhau và vốn có chứa bất kỳ ký tự nào được tìm thấy trên bàn phím của máy tính. Bất kỳ ký tự nào trong số những ký tự ở đây cũng phải được diễn dịch sang ngôn ngữ máy. Ví dụ lúc bạn gõ nhập mẫu tự "T" hoặc nhấn thanh cắt, thì thông điệp của bạn phải được trình bày trong bộ nhớ của máy tính theo một số cách. Một mã chuẩn phát triển với một cách trình bày dạng số đặc biệt ứng với mỗi một ký tự được các ngôn ngữ lập trình sử dụng. Mã tiêu chuẩn hóa thì dành để trao đổi thông tin (ASCII) được sử dụng một cách nhất quán bởi hầu hết các máy tính và thiết bị ngoại vi. Những con số từ 0 cho đến 255 biểu thị mỗi mẫu tự, số, khoảng trống, dấu chấm câu và ký tự khác có sẵn cho các ngôn ngữ lập trình. Phần sơ đồ ASCII thường được dùng được cho trong bản 3.1. ASCII 0 cho đến 31 và 127 là các ký tự không thể in nó biểu thị các thông điệp khác nhau có thể được gửi đi. Ví dụ, 12 phải gửi mẫu vào máy in và 7 tạo nên một âm thanh beep.

***Bảng 3-1** Bộ ký tự ASCII*

Decimal	Character	Decimal	Character	Decimal	Character	Decimal	Character
0	nul	32	space	64	@	96	`
1	soh	33	!	65	A	97	a
2	stx	34	"	66	B	98	b
3	etx	35	#	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g

8	bs	40	(	72	H	104	h
9	ht	41	)	73	I	105	i
10	nl	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	soh	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	p
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w
24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

CHỦ ĐIỂM 3.2

VARIABLES AND CONSTANTS
Các biến và hằng

A variable is a memory location whose contents can be filled and changed during the execution of the program. A *constant* is a memory location whose contents stay the same during the execution of the program. The computer needs to know which locations will be accessed during the program, so all variables and constants must be declared in some way. Declaring variables or constants entails telling the computer not only whether they will change or stay the same, but also *how many* locations will be used, *what* they will be called in the program, and the *type* of data they will hold.

3.2.1 Simple Data Types

Remember that a memory location contains a series of 1's and 0's. That series, as explained in Section 1.4, can represent a variety of things: an integer, a real number (also called a floating-point number), an ASCII code for a character, or simply a 1-bit yes/no value. For example, the 2-byte word

[00000000 01000001]

could represent the number 65 if it is an integer, or the letter "A" if it is a character. Each programming language uses different conventions. The available simple data types and the amount of memory used by each are listed in Table 3-2 for Visual Basic, C, C++, and Java. Notice that Visual Basic has no type for an individual character. All character data are processed as Strings. See Chapter 6 for a complete explanation of the VB String type.

Table 3-2 Simple Data Types.

Type	Size and Name in Visual Basic	Size and Name in C/C++	Size and Name in Java
boolean	1 bit – Boolean	Only available in some compilers	1 bit – boolean
character	no character type	1 byte – char	2 bytes – char
short integer	no short type	2 bytes – short	2 bytes – short
integer	2 bytes – Integer	4 bytes – int	4 bytes – int
long integer	4 bytes – Long	4 or 8 bytes – long	8 bytes – long
floating point	4 bytes – Single	4 bytes – float	4 bytes – float
double precision	8 bytes – Double	8 bytes – double	8 bytes – double

Before using any memory location, the program must include a declaration specifying whether the memory location represents a variable or a constant, what type of data it will contain, and what it will be called in the

program. The computer then finds a memory location and reserves that space for the whole time the program is running. The programmer only uses the name specified, not the actual address of the variable in memory. However, in C and C++ the actual memory location can be determined by use of the *address* operator, &. For example, if an integer variable *num1* is declared, its address in memory is *&num1*. Also, the number of bytes actually used can be accessed using the function *sizeof(variable)*. See Exercise 3.4 at the end of the chapter for an example.

Declaring variables in Visual Basic makes use of the Dim operator, giving the *Dimension* of the variable. The actual syntax is:

Dim *variablename* **As** *type*

In C, C++ and Java the type name is listed first, and then the variable name. The syntax is:

type variablename;

In all four languages, a constant can be declared. C, C++, and Visual Basic use the reserved word const and Java uses the word **final.** Some examples of declarations are given in Table 3-3. Notice that any programming statement in C, C++, and Java is followed by a semicolon. C, C++, and Java are case-sensitive and all data types are specified in lowercase. Although Visual Basic is not case-sensitive, the editor adjusts the Const, Dim and data type names to begin with uppercase letters.

Table 3-3 Declaring Variables and Constants.

Visual Basic	Java	C/C++	Explanation
Const NUM = 2	final int NUM = 2;	const int NUM = 2;	Declare a constant integer with the value 2.
Const PI = 3.14	final float PI = 3.14;	const float PI = 3.14;	Declare a constant floating-point number for π
Dim sum As Integer	int sum;	int sum;	Declare an integer variable called sum. No value is yet assigned to it.
Dim myCh As String	char myCh;	char myCh;	Declare a variable that will contain only one character. In Visual Basic, even one character is called a String.
Dim interestRate As Single	float interestRate;	float interestRate;	Declare a variable that will contain a floating-point number with decimal values.

Choosing the identifier name for variables and constants is left to the programmer. A convention used by most programmers uses all uppercase letters for the names of constants (e.g., PI). Variable names generally be-

gin with lowercase, but mixed case is often used for clarity (e.g., interestRate). Most languages have a few rules:

- No reserved words in the language can be used. (Reserved words are any words used to implement specific features.)
- Identifiers begin with a letter and may contain only letters, underscores, or digits.
- Identifiers are usually kept to under 30 characters in length.

The allowed length is different for each language and even sometimes for individual compilers of the same language. The general rule is that identifiers should be long enough to be meaningful, but not too long that mistakes would be made in typing. One-character variable names are not usually explicit enough to help the programmer. For example, the purpose of a variable called *sum* is more evident than a variable called x. Reading and debugging the program is much easier when variables and constants have meaningful names.

Constants are usually easy to name because they represent a specific value. Some good examples are PI, RATE, MAXNAMES. Variable names are often harder to choose. Here is a list of good variable names that are legal and also meaningful:

first_initial	num1	amtToDeposit
mpg	myList	newBalance
gallonsUsed	tempNum	side4

Remember that C, C+ +, and Java are case-sensitive. Therefore, myList would be a different variable from mylist. The programmer must be very careful to use consistent spelling and case for all identifiers.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.2

3.2 CÁC BIẾN VÀ CÁC HẰNG

Một biến là một vị trí trong bộ nhớ mà nội dung của nó có thể được lấp đầy và bị thay đổi trong suốt quá trình thực thi chương trình. Một hằng là một vị trí trong bộ nhớ mà nội dung của nó giữ không đổi trong suốt quá trình thực thi chương trình. Máy tính cần phải biết vị trí nào sẽ được truy cập trong suốt chương trình vì thế tất cả các biến và các hằng sẽ được khai báo theo một số cách. Việc khai báo các biến hoặc các hằng chi tiết hóa kể cho máy tính không chỉ là trường hợp chúng có bị thay đổi hay vẫn giữ cố định mà còn báo cho biết có bao nhiêu vị trí được dùng, chúng sẽ được gọi là gì trong chương trình và dữ liệu mà chúng giữ.

3.2.1 Các kiểu dữ liệu đơn giản

Hãy nhớ rằng mỗi vị trí bộ nhớ có chứa một chuỗi các chữ số 1 và chữ số 0. Chuỗi đó như được giải thích trong phần 1.4 có thể trình bày nhiều thứ một số nguyên, một số thực (còn được gọi là một số chấm động) một mã ASCII dành cho một ký tự, hoặc đơn giản chỉ là một giá trị yes/no 1 bit. Ví dụ, từ 2 byte chẳng hạn:

[00000000 01000001]

có thể trình bày 65 nếu nó là một số nguyên hoặc mẫu tự A nếu nó là một ký tự. Mỗi ngôn ngữ lập trình sử dụng các quy ước khác nhau. Các kiểu dữ liệu đơn giản có sẵn và lượng bộ nhớ được dùng sẽ được liệt kê trong bảng 3.2 dành cho Visual Basic, C, C++ và Java. Lưu ý rằng Visual Basic không có kiểu dành cho một ký tự riêng biệt. Tất cả dữ liệu ký tự phải được xử lý dưới dạng các String (chuỗi). Xem chương 6 để biết được phần giải thích hoàn chỉnh về kiểu VB String.

***Bảng 3-2** Các kiểu dữ liệu đơn giản*

Type	Size and Name in Visual Basic	Size and Name in C/C++	Size and Name in Java
boolean	1 bit – Boolean	Only available in some compilers	1 bit – boolean
character	no character type	1 byte – char	2 bytes – char
short integer	no short type	2 bytes – short	2 bytes – short
integer	2 bytes – Integer	4 bytes – int	4 bytes – int
long integer	4 bytes – Long	4 or 8 bytes – long	8 bytes – long
floating point	4 bytes – Single	4 bytes – float	4 bytes – float
double precision	8 bytes – Double	8 bytes – double	8 bytes – double

Trước khi sử dụng bất kỳ vị trí bộ nhớ nào, chương trình phải chứa một phần khai báo chuyên biệt hóa cho biết vị trí bộ nhớ đang trình bày một biết hay là hằng. Kiểu dữ liệu nào nó chứa, và những gì sẽ được gọi trong chương trình. Sau đó máy tính tìm một vị trí bộ nhớ đảo ngược và giữ lại chỗ đó trong suốt toàn bộ thời gian chương trình đang chạy. Nhà lập trình chỉ sử dụng tên được chỉ định, không sử dụng địa chỉ thực tế dành cho biến trong bộ nhớ. Tuy nhiên, trong C và C++ vị trí bộ nhớ thực tế có thể được xác định bằng cách sử dụng address, &. Ví dụ nếu một biến nguyên là num1 được khai báo thì vị trí của nó trong bộ nhớ là &num1. Cũng vậy số các byte thực sự được dùng có thể được truy cập bằng cách sử dụng hàm sizeo(variable). Xem bài tập 3.4 ở cuối chương này để tìm hiểu ví dụ.

Việc khai báo biến trong Visual Basic đưa đến sử dụng toán từ Dim, cung cấp kích thước của biến, cú pháp là:

Dim `variablename` **As** `type`

Trong C, C++ và Java tên kiểu được liệt kê trước tiên sau đó là tên biến. Cú pháp là:

type variablename;

Trong tất cả 4 ngôn ngữ, có một hằng cần phải được khai báo. C, C++ và Visual Basic sử dụng từ được giữ lại là const và Java sử dụng từ final. Một vài ví dụ về việc khai báo được cho trong bảng 3-3. Lưu ý rằng bất cứ câu lệnh chương trình nào trong C, C++ và Java đều phải đi theo sau một dấu (;). C. C++ và Java thường nhạy kiểu chữ in hoa hoặc in thường, tất cả các dữ liệu được chỉ định theo kiểu chữ thường. Mặc dù, Visual Basic không nhạy kiểu chữ nhưng trình biên soạn Const, Dim và tên kiểu dữ liệu bắt đầu bằng các mẫu tự hoa.

***Bảng 3-3**. Khai báo các biến và các hằng*

Visual Basic	Java	C/C++	Explanation
Const NUM = 2	final int NUM = 2;	const int NUM = 2;	Declare a constant integer with the value 2.
Const PI = 3.14	final float PI = 3.14;	const float PI = 3.14;	Declare a constant floating-point number for π.
Dim sum As Integer	int sum;	int sum;	Declare an integer variable called sum. No value is yet assigned to it.
Dim myCh As String	char myCh;	char myCh;	Declare a variable that will contain only one character. In Visual Basic, even one character is called a String.
Dim interestRate As Single	float interestRate;	float interestRate;	Declare a variable that will contain a floating-point number with decimal values.

Việc chọn tên của bộ nhận dạng dành cho các biến và các hằng là quyền của nhà lập trình. Có một quy ước được tất cả hầu hết các nhà lập trình thường dùng đó là sử dụng tất cả các mẫu tự in hoa dành cho tên các hằng (ví dụ PI). Các tên biến thường bắt đầu bằng chữ thường, các kiểu chữ hoa và thường hòa trộn với nhau được dùng để làm nổi bật (ví dụ interrestRate). Hầu hết các ngôn ngữ đều có một quy tắc sau đây:

- *Không có từ giữ lại nào trong ngôn ngữ có thể được dùng (các từ giữ lại là bất cứ từ nào được dùng để thực thi các tính năng đặc biệt).*
- *Các công cụ nhận dạng bắt đầu bằng một mẫu tự và có thể chỉ chứa một vài mẫu tự dấu gạch dưới hoặc chữ số.*
- *Các công cụ nhận dạng thường được giữ có chiều dài dưới 50 ký tự.*

Chiều dài được cho phép thì khác nhau tùy theo từng ngôn ngữ và thậm chí đôi khi đối với các trình biên soạn riêng biệt của một ngôn ngữ. Quy tắc chung đó là các bộ nhận dạng cần phải có chiều dài đủ để mang ý nghĩa nhưng không quá dài đến nỗi phải gây ra các sai lỗi khi gõ nhập. Các tên biến một ký tự thường không đủ hiển nhiên để giúp nhà lập trình. Ví ậu mục đích của một biến có tên là sum thì mang tính thuyết phục hơn là biến có tên là x. Việc đọc và gỡ rối một chương trình dễ dàng hơn nhiều các biến và các hằng có các tên mang đầy đủ ý nghĩa.

Các hằng thường dễ đặt tên bởi vì chúng biểu thị cho một giá trị đặc biệt. Một vài ví dụ về chúng là PI, RATE, MAXNAME. Các tên biến thường khó chọn hơn. dưới đây là danh sách các tên biến thường hợp pháp và có ý nghĩa:

first_initial	**num1**	**amtToDeposit**
mpg	**myList**	**newBalance**
gallonsUsed	**tempNum**	**side4**

Hãy nhớ rằng C, C++ và Java đều là nhạy kiểu chữ. Do đó myList sẽ có một biến khác với mylist. Nhà lập trình họ rất cần thận để sử dụng kiểu viết và kiểu chữ in và chữ thường nhất quán cho tất cả các bộ lệnh nhận dạng.

CHỦ ĐIỂM 3.3

ASSIGNMENT STATEMENTS
Các lệnh gán

Once you have chosen a name and reserved the location for a variable, you must give it a value. There are three main methods of getting values into variables:

- Read in a value from a file
- Ask the user to type in a value
- Assign a value within the program

File input and output will be covered in Chapter 7. Reading in values from the keyboard will be covered later in this chapter. This section explains ***assignment*** statements, or statements that explicitly place values into memory locations. The syntax for an assignment statement is:

variable name = value

The assignment is always made from left to right. There are a few rules for assignment statements that must be followed in every language.

(1) Only one variable name can be on the left of the equal sign because it is the destination location that will be changed.
(2) Constants cannot be on the left, because they cannot be changed.
(3) The value on the right of the equal sign may be a constant, another variable, or a formula or arithmetic expression combining constants and variables.
(4) Anything on the right is not changed.
(5) The variable and value must be of the same data type.

EXAMPLE 3.1 Under the column **Statements** below, there is a series of GC++ assignment statements that are to be executed in the order indicated. What is the content of each variable after each statement has been executed, knowing that *areaOfSquare* has been declared a floating-point variable, *initial* is a character, and *length* is an integer?

Statements	Contents of the Variables after the Statement		
	initial	*length*	*areaOfSquare*
a. before anything is done:	undefined	undefined	undefined
b. length = 5;	undefined	5	undefined
c. initial = 'P';	'P'	5	undefined
d. areaOfSquare = length;	'P'	5	5.0
e. areaOfSquare = initial;	'P'	5	80.0
f. areaOfSquare = 37.5;	'P'	5	37.5
g. length = areaOfSquare;	'P'	37	37.5
h. initial = areaOfSquare;	'%'	37	37.5

Lines b, c, and f are simple assignments placing constant values into the variables. Line d allows an integer value to be placed into a floating-point variable, and line a allows the ASCII value of the character "P" to be placed into the floating-point variable. Nothing is lost in either line. However, in lines g and h, a floating-point value was placed into a character and an integer variable. Although C/C++ gives no error messages, the floating-point constant will be truncated when placed into the integer variable. Likewise, the same integer part will be interpreted as an ASCII value when assigned to the character variable.

These statements would produce similar results in Java for lines b through L However, Java would not allow lines g or h to compile. In that language it is illegal to assign a floating-point value to an integer or character variable.

EXAMPLE 3.2 Under the column **Statements** below, there is a series of Visual Basic assignment statements that are to be executed in the order indicated. What is the content of each variable after each statement has been executed, knowing *areaOfSquare* has been declared a single variable, *initial* as a string, and *length* as an integer?

Statements	Contents of the Variables after the Statement		
	initial	*length*	*areaOfSquare*
a. before anything is done:	space	0	0
b. length = 5	undefined	5	undefined
c. initial = "P"	"P"	5	undefined
d. areaOfSquare = length	"P"	5	5.0
e. areaOfSquare = initial	Gives an error message - type mismatch		
f. areaOfSquare = 37.5	"P"	5	37.5
g. length = areaOfSquare	"P"	38	37.5
h. initial = areaOfSquare	"37.5"	37	37.5

Notice that in Visual Basic, variables have default values before anything is assigned to them. In line g, assigning a floating-point number to an integer variable rounds the floating value to the nearest integer value rather than truncating. In line h, assigning a floating-point number into a string actually assigns the string value of the number. Also, C and C++ require single quotation marks around a single character. Visual Basic treats a single character as if it were a string of length 1, and requires double quotation marks.

Both these examples show that one must be very careful when writing assignment statements. Mixing data types on the left and right sides of assignment statements is very dangerous, and may produce unpredictable results. It is important always to follow the rule of explicitly assigning values of the same data type as the variable that appears on the left-hand side of an assignment statement.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.3

3.3 CÁC CÂU LỆNH GÁN

Một khi bạn đã chọn một tên và đã dành vị trí cho một biến, bạn phải cho là một giá trị. có ba phương pháp chính để đưa giá trị vào biến:

- *Đọc một giá trị từ một file yêu cầu*
- *Người dùng gõ nhập một giá trị*
- *Gán một giá trị bên trong chương trình*

Các file nhập và xuất được trình bày chương 7. Việc dọc các giá trị từ bàn phím sẽ được trình bày phần sau trong chương này. Còn một lời giải thích các câu lệnh gán, hoặc tất cả các lệnh nhằm đưa các giá trị và các vị trí của bộ nhớ một cách rõ rệt. Cú pháp của một câu lệnh gán là

Variable name = value

Vì các mã luôn luôn được hiển thị từ trái sang phải. Có một số quy tắc dùng cho các câu lệnh gán sẽ được tuân thủ trong mỗi một ngôn ngữ.

(1) Chỉ có một tên biến có thể nằm phía bên trái của dầu bằng bởi vì nó là vị trí đích nên được thay đổi.

(2) Các hằng không thể nằm phía bên trái bởi vì chúng không thể bị thay đổi,

(3) Giá trị nằm phía bên phải của dấu bằng có thể là một hằng, một biến khác hoặc một công thức hoặc một biểu thức số học kết hợp với các hằng và các biến.

(4) Bất cứ nội dung nào nằm phía bên phải thì không được thay đổi.

(5) Biến và giá trị phải có cùng kiểu dữ liệu.

VÍ DỤ 3.1 *Bên dưới câu lệnh Statements cột dưới đây, có một chuỗi các câu lệnh gán C/ C++ được thực thi theo một thứ tự chỉ định, nội dung của mỗi một biến sau mỗi câu lệnh phải được thực thi là gì, biết rằng areaOfSquare được khai báo với một biến chấm động, khởi tạo là một ký tự và chiều dài là một số nguyên?*

Statements	Contents of the Variables after the Statement		
	initial	*length*	*areaOfSquare*
a. before anything is done:	undefined	undefined	undefined
b. length = 5;	undefined	5	undefined
c. initial = 'P';	'P'	5	undefined
d. areaOfSquare = length;	'P'	5	5.0
e. areaOfSquare = initial;	'P'	5	80.0
f. areaOfSquare = 37.5;	'P'	5	37.5
g. length = areaOfSquare;	'P'	37	37.5
h. initial = areaOfSquare;	'%'	37	37.5

Các dòng b, c và f là các câu lệnh gán đơn giản đặt các giá trị hằng số và các biến. Dòng d cho phép giá trị nguyên sẽ được đặt vào trong biến chấm động, và dòng e cho phép giá trị ASCII của ký tự p được đặt vào trong biến dấu chấm động. Không có gì bị mất trong mỗi dòng. Tuy nhiên, trong các dòng g và h, một giá trị dấu chấm động được đặt vào trong một ký tự và một biến nguyên. Mặc dù C, C++ không cho ta thông điệp thông báo lỗi nhưng hằng số chấm động sẽ bị cắt tỉa lúc được đặt vào trong biến nguyên. Cũng vậy, phần nguyên giống nhau sẽ được diễn dịch dưới dạng một giá trị ASCII lúc được gán vào biến ký tự.

Những câu lệnh này sẽ tạo ra các kết quả giống nhau trong Java dành cho các dòng b cho đến f. Tuy nhiên, Java sẽ không cho phép các dòng g goặc f biên soạn. Trong ngôn ngữ đó, điều không hợp lý đó là bạn gán một giá trị chấm động và một biến nguyên hoặc một biến ký tự.

VÍ DỤ 3.2 *Sử dụng cột Statements có một chuỗi các câu lệnh gán Visual Basic được thực thi theo thứ tự chỉ định. Nội dung của mỗi một biến sau mỗi câu lệnh sẽ được thực thi là gì, biết rằng areaOfSquare được khai báo là một biến đơn, khởi tạo dưới dạng một chuỗi và chiều dài là một số nguyên.*

Statements	Contents of the Variables after the Statement		
	initial	*length*	*areaOfSquare*
a. before anything is done:	space	0	0
b. length = 5	undefined	5	undefined
c. initial = "P"	"P"	5	undefined
d. areaOfSquare = length	"P"	5	5.0
e. areaOfSquare = initial	Gives an error message - type mismatch		
f. areaOfSquare = 37.5	"P"	5	37.5
g. length = areaOfSquare	"P"	38	37.5
h. initial = areaOfSquare	"37.5"	37	37.5

Lưu ý rằng trong Visual Basic có các biến các giá trị về mặc định trước khi bất cứ nội dung nào được gán cho chúng. Trong dòng g, gán một số chấm động vào một biến nguyên được làm tròn sang giá trị động nằm gần giá trị duy nhất thay vì sẽ được làm tròn sang giá trị động nằm gần với giá trị duy nhất hơn là cắt xén nó. trong dòng h việc gán một số chấm động vào một chuỗi thật sự là gán giá trị chuỗi của số đó. Cũng vậy, C và C++ yêu cầu phải có một dấu bao quanh ký tự đơn. Visual Basic xử lý ký tự đơn y hệt như chuỗi có chiều dài bằng 1 yêu cầu phải có dấu móc kép.

Cả hai ví dụ này đều cho thấy rằng người ta phải rất thận trọng lúc viết các câu lệnh gán. Việc trộn các kiểu dữ liệu ở phía bên trái và bên phải của câu lệnh gán rất là nguy hiểm và có thể tạo ra các kết quả ngoài ý muốn, điều quan trọng là phải luôn luôn tuân thủ quy tắc về việc gán các giá trị của cùng kiểu dữ liệu dưới dạng các biến xuất hiện ở phía bên trái của câu lệnh gán.

CHỦ ĐIỂM 3.4

ARITTHHMETIC EXPRESSIONS AND OPERATOR PRECEDENCE

Các biểu thức số học và sự ưu tiên của toán tử

The examples above placed only simple values, either constants or single variables, into the destination variables. The power of assignment statements is made possible through the use of formulas and arithmetic expressions. The expression on the right is computed and then the result is assigned to the destination variable on the left.

The arithmetic operators used in expressions are listed in Table 3-4. The order in which the operations are executed follows the rules of arithmetic, with items in parentheses executed first, then multiplication, division, and modulus, and finally addition and subtraction. Operations with the same order of precedence are always computed left to right.

C, C++, and Java do not have explicit exponentiation operators. Exponentiation must be computed explicitly, or through the use of library functions, which will be explained later.

Table 3.4 Order of Precedence of Arithmetic Operators.

Operators	Operation	Order of Precedence	Associativity
( )	Parentheses	1	Inside out
^ (VB only)	Exponentiation	2	
* /	Multiplication and division	3	Left to right
\ (VB only)	Integer division	5	Left to right
% (C, C++, Java) or MOD (VB)	Modulo arithmetic	5	Left to right
+ −	Addition and subtraction	6	Left to right

A special note is important regarding division. Programming languages treat integer division differently from floating-point division. Division performed with floating-point numbers as either the dividend or the divisor will behave as expected, resulting in quotients with fractional parts. If both are integers, then integer division is performed instead, and the quotient is truncated to only the integer part of the value. Here are a few C/C++ examples where real1 and real2 have been declared as floating-point numbers.

```
real1=5.6;
real2=real1/2;      //floating point divided by integer --
                      real2 contains 2.8
```

```
real2=25/2.0;        //integer divided by floating point --
                       real2 contains 12.5
real2=25/2;          //integer divided by integer -- real2
                       contains 12
```

In Visual Basic integer division is done explicitly by changing the integer division sign from "/" to "\", as shown below.

```
real1=5.6
real2=real1/2        ' single dimension divided by integer
                       --- real2 contains 2.8
real2=25/2           ' integer divided by integer using regu-
                       lar division operator
                     ' /-- real2 contains 12.5
real2=25\2           ' integer divided by integer using in-
                       teger division operator
                     ' \-- real2 contains 12
```

In integer division, the division results in the truncated quotient. To find the remainder, the MOD or % operator is used. Consider the examples in Fig. 3-1 where 26 divided by 3 results in a quotient of 8, with a remainder of 2.


```
                    8  <= quotient
                 3|26
                  24
                   2 <=remainder
C++ example:
     firstInt=26;
     secondInt=firstInt/3;      //secondInt contains 8
     thirdInt=firstInt % 3;     //thirdInt contains 2
VB example:
     firstInt=26
     secondInt=firstInt\3       ' secondInt contains 8
     thirdInt=firstInt Mod 3    ' thirdInt contains 2
```

Fig. 3-1 Modulo arithmetic.

EXAMPLE 3.3 Write the C/C++ and Visual Basic assignment statements to calculate the miles per gallon of a vehicle. Assume *distance*, *gallonsUsed*, and *mpg* have been declared as floating-point numbers, and *distance* and *gallonsUsed* have values '

```
mpg=distance / gallonsUsed;       // C/C++ statement
mpg=distance / gallonsUsed ' VB statement
```

EXAMPLE 3.4 Write the C/C++ and Visual Basic assignment statements to change a Fahrenheit temperature into Celsius. Assume *fahrenheit* and *celsius* have been declared as floating-point numbers, and *fahrenheit* has a value.

```
celsius=(5.0/9)*(fahrenheit-32);
                    //C/C++ -- notice either the dividend or
                    //divisor MUST be floating point
celsius=(5/9)*(fahrenheit-32)
                    ' vB statement -/ is used to force
                    ' floating-point response
```

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.4

3.4 CÁC BIỂU THỨC SỐ HỌC VÀ TRÌNH TỰ TOÁN TỬ

Ví dụ trên đây chỉ đưa ra các giá trị đơn hoặc các hằng hoặc các biến vào các biến đích. Sức mạnh của các câu lệnh gán là nó có thể sử dụng các công thức và công thức đại số. Biểu thức nằm phía bên phải được tính toán rồi gán vào biến đích nằm phía bên trái.

Các toán tử số học được dùng trong các biểu thức sẽ được liệt kê trong bảng 3-4. thứ tự mà qua đó toán tử thực thi tuân theo các quy tắc của số học với các số hạng được đặt trong dấu ngoặc đơn sẽ được thực thi trước. Sau đó là phép nhân, phép chia và module cuối cùng là phép cộng và phép trừ. Các toán tử có thứ tự ưu tiên giống nhau thì luôn luôn được tính từ trái sang phải.

C, C++ và Java không có các toán tử số mũ rõ ràng. Các toán tử số mũ phải được tính một cách hiển nhiên hoặc thông qua việc sử dụng các hàm thư viện sẽ được giải thích rõ ở phần sau.

***Bảng 3-4** Thứ tự ưu tiên của các toán tử số học*

Operators	Operation	Order of Precedence	Associativity
( )	Parentheses	1	Inside out
^ (VB only)	Exponentiation	2	
* /	Multiplication and division	3	Left to right
\ (VB only)	Integer division	5	Left to right
% (C, C++, Java) or MOD (VB)	Modulo arithmetic	5	Left to right
+ −	Addition and subtraction	6	Left to right

VÍ DỤ 3-3 *Hãy viết các câu lệnh gán C, C++ và Visual Basic để tính số dặm đi được trện mỗi gallon của một chiếc xe hơi. Giả sử distance, galosUsed và mpg đã được khai báo dưới dạng các dấu chấm động. Distance và gallonsUsed có các giá trị.*

```
mpg=distance / gallonsUsed;       // C/C++ statement
mpg=distance / gallonsUsed ' VB statement
```

VÍ DỤ 3.4

Hãy viết các câu lệnh gán C, C++ và Visual Basic để thay đổi nhiệt độ Fahrenheit sang Celsius. Giả sử fahrenheit và celsius được khai báo dưới dạng số chấm động và Fahrenheit có một giá trị.

```
celsius=(5.0/9)*(fahrenheit-32);
                    //C/C++ -- notice either the dividend or
                    //divisor MUST be floating point
celsius=(5/9)*(fahrenheit-32)
                    ' vB statement -/ is used to force
                    ' floating-point response
```

CHỦ ĐIỂM 3.5

COMMENT STATEMENTS
Các câu lệnh chú thích

Programming languages provide a way for programmers to embed in the code explanatory statements, or comments, that are non-executable. These statements are ignored by the compiler, but provide explanation for the human beings who are reading or writing the code. This is **called documentation.**

Comments are critical to the understanding of code, and should be used liberally. There are a few general rules for including comments in the code:

(1) Include comments at the beginning listing the program name, the programmer's name, and a complete explanation of what the program is supposed to do.

(2) Include comments explaining each separate part of the code.

(3) Include comments explaining formulas or complex expressions.

(4) Include comments anywhere else they might be of help.

The method of commenting is different for each language. Visual Basic uses the apostrophe and C, C++, and Java use the double slash. Both indicate that everything else on the line should be ignored by the compiler. See Examples 3.3 and 3.4 above.

C, C++, and Java also allow commenting multiple lines. Here is an example:

```
/*The slash and star begin this multiple line comment which
  can be used for long explanations. It will be ignored by the
  compiler until the final star and slash are found. */

... rest of program
```

Comments will frequently be used in this book for explanatory purposes. Many programmers feel that you can never use too many comments.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.5

3.5 CÁC CÂU LỆNH BÌNH CHÚ

Ngôn ngữ lập trình cung cấp một cách để cho các nhà lập trình đính kèm trong mã các câu lệnh giải thích hoặc câu bình chú vốn không thể thực thi. Những câu lệnh này bị trình biên soạn bỏ qua nhưng nó cung cấp phần giải thích của người đang đọc hoặc viết mã. Đây được gọi là tài liệu.

Các lời bình chú là chuẩn mực để hiểu được mã, và nên được sử dụng một cách rộng rãi. Có một số quy tắc chung để đưa vào các câu bình chú trong mã.

(1) Đưa các câu bình chú ở phần bắt đầu của danh sách tên chương trình, tên nhà lập trình, phần giải thích hoàn chỉnh về những gì mà chương trình đề nghị phải làm.

(2) Đưa vào các lời bình chú đang giải thích các phần riêng biệt của mã.

(3) Đưa vào các lời bình chú giải thích các công thức của các biểu thức phức.

(4) Đưa vào các lời bình chú vào bất cứ nơi nào khác có thể sẽ giúp được.

Phương pháp đưa lời bình chú khác nhau tùy theo mỗi một ngôn ngữ. Visual Basic thì sử dụng dấu apostrophe còn C, C++ và Java thì sử dụng dấu double slash. Cả hai đều chỉ định rằng mỗi một cú pháp nằm trên dòng cần phải được trình biên soạn bỏ qua. Xem ví dụ 3.3 và 3.4 trên đây.

C, C++ và Java cũng cho phép có nhiều dòng bình chú. Dưới đây là một ví dụ

```
/*The slash and star begin this multiple line comment which
  can be used for long explanations. It will be ignored by
  the compiler until the final star and slash are found. */
... rest of program
```

Các lời bình chú thường được dùng trong sách này để giải thích mục đích. Nhiều nhà lập trình cảm thấy rằng bạn không bao giờ sử dụng quá nhiều bình chú.

CHỦ ĐIỂM 3.6

SIMPLE INPUT/OUTPUT
Nhập/xuất đơn giản

Recall that the basic flow of computer processing is input, process, output. Programs can read input from and write output to files on disks. Handling of file input/output will be covered in Chapter 7. For the purposes of the next few chapters, we will assume there is a user sitting at the computer to provide the input from the keyboard and wanting to see the output on the screen. The program itself is supposed to process the information given by the user and then display the output. There must be program statements to accomplish these tasks.

One of the biggest differences between computer languages is how they handle input and output. Therefore, each of these languages will be addressed separately. Also, within each language there may be a variety of methods for input and output. Only the simplest will be addressed here.

3.6.1 C++ I/O

C++ does not have any built-in input/output commands. All C++ compilers, however, contain a package of object-oriented classes, **called the iostream** classes. The compiler views everything going in and out as streams of data. The program must extract the information from the input stream object, and insert data into the output stream object. The streams are viewed as separate objects, and the package is **called a library** of functions to help the programmer with the extraction (istream) and insertion (ostream) processes. The programmer must specify that these functions will be used by giving the compiler a message to include the required library. The syntax to include this library is:

```
#include <iostream.h>
```

The command must be given first, before any other program statements, and the # must be in the first column of the line. The compiler then finds the header file (.h) that explains all the resources that can be used in the program. A few of these are listed with explanation in Table 3-5.

The function for input is cin (pronounced "see in" for ConsoleINput). The extraction operator is available for use with any built-in data type. Whatever is typed at the keyboard is examined to be sure it is the correct data type, and then placed into the specified variable. A character variable will take the ASCII value of whatever is typed. A numeric value will accept only a number. If anything else is typed, the program crashes with a run-time error of data type mismatch.

Table 3-5 Some iostream.b Library Resources

Resourc	Type	Explanation
cin	object	Controls extraction from standard stream of bytes coming from keyboard
>>	extraction operator	Gets bytes from an input stream object
cout	object	Controls insertion to the standard stream of bytes going out to the screen
<<	insertion operator	Inserts bytes into the standard output stream object
cerr	object	Controls immediate insertion of messages to the standard streamof bytes going out to the screen
endl or '\in'	constant manipulator	Inserts newline character into the output stream object makes the cursor go to the next line

EXAMPLE 3.5 The following example illustrates the use of cin.

General Syntax: `cin>>variable;`

Examples:

```
cin>>num1;          // if a number is typed its value is placed
                       into num1

cin>>char2;         // whatever is typed its ASCII value is
                       placed into chart

cin>>num1>>num2;    // two numbers must be typed, separated
                       by a space
```

The function for output is cout (pronounced "see out" for ConsoleOUTput). Often another command is used for error messages, cerr (pronounced "see err" for Console ERRor messages). These messages are also displayed on the screen. However, it is helpful for the programmer to distinguish regular output from specific error messages that must be given by the program as it is executing. The insertion operator can be used with any built-in data type. If the variable is a numeric value, it is first converted into its ASCII representation in order to be displayed on the screen. Any number of items may be inserted into the output stream object, including the constant **endl** which sends the cursor to the next line. A string of characters contained in quotation marks can also be displayed.

EXAMPLE 3.6 The following illustrates the use of cout.

General Syntax: cout<<*output*;

Where *output* is a variable, a constant, "a literal string", or an expression

Statements where numl is 4 and charl is 'p':	**Output:**
`cout<<numl ;`	4
`cout<<"The number is "<<numl<<endl;`	The number is 4
`cout<<"Your initial is "<<charl<<endl;`	Your initial is p
`cout<<"line 1"<<endl<<"line 2"<<endl;`	line 1 line 2
`cout<<"The sum of 4 and 5 is "<<4+5<<endl;`	The sum of 4 and 5 is 9

The programmer often needs to combine output and input in order to give a message to the user as to what should be typed on the keyboard. Never assume the user knows what to type. For example, the C++ lines needed to ask the user to type a positive integer and then read it into the variable poslnt would be:

```
cout<< "Enter a number greater than zero =>";
cin>>posInt;
```

This concept of offering prompts to the user is important in any language. Directions should be clear and specific in order to minimize run-time crashes of type mismatch. In reality, the program should always check the input and allow for giving messages regarding incorrect data.

3.6.2 Java I/O

As specified at the beginning of this chapter, a Java program can be a stand-alone application or an applet that is embedded in a Web page. Input and output are handled very differently for those two situations. This section will present only the simplest input and output needed for a regular Java application. The Java I/O library must be included, or imported, in order to accomplish this task. The syntax for this is shown below. The asterisk means that ALL the functions and objects available in the javado library will be imported.

```
import java.io.*;
```

Java uses the System object to read and print the stream object. Table 3-6 shows the most common I/O functions used in Java.

Like C++, Java uses one object to read streams of data coming in from the keyboard, Systemin, and another to print output streams to the screen, **System.out.** Java (and also C++) has a temporary memory location to hold input and output which is called the *buffer.* If the program has been giving output, the buffer must be cleared, or flushed, before trying to read input. System.out.flush() is the function needed to be sure the buffer is clear.

Table 3-6 Java I/O Commands.

Command	Explanation
System.out.print(..)	Insertion of whatever is in the parentheses into the standard stream of bytes going out to the screen
System.out.println(..)	Insertion of whatever is in the parentheses into the standard stream of bytes going out to the screen, then inserts newline character
System.out.flush()	Forces whatever is in the buffer to be displayed before reading
System.in.read()	Controls extraction from standard stream of bytes coming from keyboard
System.in.skip(n)	Skips n characters of input stream - used to pass unwanted values such as Enter

Java System input catches one character at a time and assumes it is a character even if we are looking for integers. Therefore, we must specifically change the incoming character to an integer. Remember that the ASCII value of "0" is 48, of "1" is 49, and so forth. Therefore, we can subtract the ASCII value of "0", or 48, from the character to change it into an integer. Example 3.7 will print 5 if 5 is entered. Without the subtraction, it would have printed 53, or the ASCII value of 5.

EXAMPLE 3.7 The following illustrates the use of Java System input and output.

```
System.out.print ("Enter a number");
                                    // give instructions to user
System.out.flush(); // flush the buffer before reading
num1=System.in.read();      // read the integer that is typed
num1=num1 - '0',            // subtract 48 to give the integer
                            // value, not ASCII
System.out.println ("Your number is"+num1);
```

It should also be noted that System.in.read() will accept only one character. If 999 is entered in the example above, only the first 9 is read. If two items are to be read, and the Enter key is pressed between them, that character must be skipped. The number of characters to skip depends upon the actual implementation. This is demonstrated in Example 3.8. Input and output are easier with Java applets, which is beyond the scope of this book.

EXAMPLE 3.8 The following illustrates more Java System input and output.

```
System. out. print ("Enter a number");
System.out.flush();
num1=System. in. read () ;  //user types integer
                            // and then presses Enter
num1=num1-'0';
System. out. print ln ("Your number is"+num1);
System. in. skip (2) ;              //skip the Enter key before
                            //reading new data
System. out. print ("Enter a number");
System.out.flush();
num1=System.in.read();
num1=num1-'0';
System. out. println ("Your number is"+num1);
```

3.6.3 Visual Basic I/O

Visual Basic I/O is not as complicated as Java and C++. Input and output are not streams of character data, but specific discrete values coming in and going out. There are several ways of managing I/O in a visual environment. VB's object-oriented environment will be discussed further in Chapter 8. In this chapter, only the simplest I/O from a code module will be explained.

The "Visual" part of Visual Basic implies that one can see specific objects on the screen. Input is accomplished through the use of Input boxes, and output through Message boxes. Syntax for these objects is shown in Table 3-7. Another output command, Print, is also described. It will be used in many VB examples in this text. In the VB environment, it prints output to the main program form, or window.

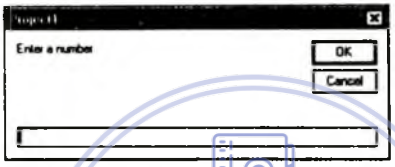
Table 3-7 Visual Basic I/O.

VB Command Syntax	Explanation
variable = InputBox(*prompt*)	prompt is message to user, and *variable* is filled with value typed into box by user
MsgBox(message)	message is displayed in a box on screen
Print *variable or constant list*	Simple print used to print to VB form
Str(value)	Converts numbers into Strings that can be displayed in boxes usually not necessary with Print
+ or &	Used to connect string parts into a long string to be displayed in boxes or with Print

Input boxes are dialog boxes, with the prompt displayed and a place for the user to respond. The user types a value into the box, and that value is placed into the variable. VB is very flexible. The input comes into the program as a string. However, if the variable destination is an integer or single precision value, the input is converted automatically into the correct data type. Example 3.9 shows examples of two InputBox commands, along with the resulting output.

EXAMPLE 3.8

```
num1=InputBox("Enter a number")
```

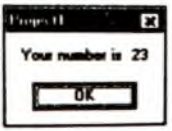



```
char 1=InputBox("Enter a character")
```

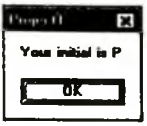
Message boxes display the message in a dialog box and then wait for the user to click OK. There are options for both Input and Message boxes to allow different buttons to be displayed. At the present, however, we will use the default values. The message sent to the Message box must be a string. If integers or floating-point numbers are to be displayed they must be converted into a string with the Str(value) function. All parts of the String message are concatenated with the plus sign or ampersand. Example 3.10 shows several examples of the Message box command with its resulting output. Notice the use of the ASCII character 10 to force the cursor to the next line. This corresponds to the *endl* of C++.

EXAMPLE 3.10

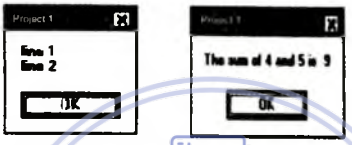
```
MsgBox ("Your number is"+Str(num1))
```



```
MsgBox ("Your initial is"+char1)
```




```
MsgBox ("line 1" & Chr(10) & "line 2")
MsgBox ("The sum of 4 and 5 is" & Str(4+5))
```



3.6.4 C I/O

The most commonly used library functions to do I/O operations in C are printf() and scanf(). The printf() function allows us to write to the standard output device, the video screen, formatted and unformatted data. Likewise, the scanf() function allows us to read values from the standard input device, the keyboard. In Chapter 7, we will use similar instructions to do file I/O. In this chapter, we will illustrate the use of these functions with several examples. The general format of these two functions is as follows:

```
printf(control string, list of variables)
scanf(control string, list of variables)
```

where control string is a string of characters, enclosed in double quotes, that may contain control specifications. Although the purpose of the control string is very similar for both functions, there are slight differences in their use. The control string in printf () generally indicates how the data are to be formatted for output. That is, it is concerned with the appearance of the data when printed. The control string in scanf() is a conversion control sequence. That is, it specifies the type of data being input by the user. The list of variables in printf() corresponds to those variables or constants that are to be printed. These variables or constants need to be separated by commas. The list of variables in scanf() corresponds to the list of variables whose values will be read in. However, unlike printf(), the scanf() function requires that the addresses of the variables rather than their names be used in the list of variables. Variables in this list also need to be separated by commas. In general, for both printf() and scanf() there is a one-to-one correspondence between the conversion characters and the

variables or constants in the list of variables. To use any of these two functions it is necessary to include the header file stdio.h.

All conversion specifications begin with a % character and end with a conversion character. Partial lists of the conversion characters most commonly used in conjunction with printf() and scanf() are shown in Tables 3-8 and 3-9 respectively.

When outputting information to a device like the video screen, in addition to the conversion characters, the control string may contain escape sequences. An escape sequence is a single character that is represented by a combination of two other characters. The first of these two characters is the "\" backslash character; the second character of the escape sequence is a letter. Table 3-10 shows a partial list of escape sequences.

Table 3-8

Conversion Character for printf()	Associated Argument is Printed
c	as character
d	as decimal
f	as floating point
s	as a string

Table 3-9

Conversion Character for scanf()	Associated Argument is Converted to
c	a character
d	a decimal
f	a floating-point number
s	a string
Lf	a double

Table 3-10

Escape Sequence	Effect
\a	make a beep sound
\b	backspace
\f	new page or form
\n	new line
\t	horizontal tab

EXAMPLE 3.11 Write the message "Hello World" to the screen.

To write messages to the screen we can use the printf() function in its simplest form. To print the given message we can use the printf() function as follows:

```
printf("Hello World");
```

This statement will print "Hello World" on the screen beginning at the current position of the cursor. That is, if we assume that the cursor is on column 10 on the screen, this instruction will print the message beginning on column 10. However, if we need to print this message on column 1 on the next line, the instruction needs to be slightly modified as follows:

```
printf("\nHello World");
```

Notice that in this case we have used the escape sequence "\n". As indicated in Table 3-10, the effect of this sequence is to "go to a new line" before printing the message.

EXAMPLE 3.12 Write the message "I am fine" beginning on a new line, five spaces to the right. Call the attention of the user by making a beep sound.

To do all this, the printf() function needs to be written as follows:

```
printf("\n\t\t I am fine\a");
```


In this statement, the effect of the escape sequences, from left to right, is as follows:

"\n" moves the cursor to a new line.

"\t" moves the cursor to the next "tab". Tabs are separated 5 spaces from each other.

After printing the message, the escape sequence "\a" makes the beep sound.

As indicated before, the printf() function can also be used to control the appearance of the output. This is generally accomplished by means of a combination of control string characters and field width specifiers. A specifier is an integer number, written between the % sign and the control letter, that indicates, among some other things, the number of output columns that are available for writing a variable or a constant. When specifiers are used, numeric values are right justified and character strings are left justified. The following example illustrates this.

EXAMPLE 3.13 What is the output of the following printf() statement?

```
printf ("The sum of %5d and %5d is %7d" , 20, 25, 45) ;
```

Within the control string there are three control characters; each one of these control characters is associated with one and only one of the con-

stants appearing in the "variable list." The association, from left to right, between the control character, the width specifier, and the constants, is as follows:

%5d is associated with 20	Write the integer 20 using a field 5 columns wide
%5d is associated with 25	Write the integer 25 using a field 5 columns wide
%7d is associated with 45	Write the integer 45 using a field 7 columns wide

Therefore, the output looks like this:

```
The sum of 20 and 25 is  45
```

Notice that all numeric values are right justified within their fields. What this implies is that the numbers are "written backwards." That is, the last digit of the number gets written in the last column of the field; the remaining digits are "written" moving toward the left. This function writes the number 20 in a field that is 5 columns wide; therefore, there are three blanks at the left of the digit 2. A similar situation occurs with the other constants.

EXAMPLE 3.14 Is there any difference between the output produced by the print statements of column A and the print statements of column B? Assume that the statements, in each column, are executed in the order indicated and independently of the statements in the other column.

Column A	Column B
`printf("\n%d+",1);`	`printf("\n%5d+",1);`
`printf("\n%d",10);`	`printf("\n%5d",10);`
`printf("\n%d",100);`	`printf("\n%5d",100);`
`printf("\n_____");`	`printf("\n_____");`
`printf("\n%d",111);`	`printf("\n%5d",111);`

Notice that the statements under column A do not have a width specifier. In this case, C uses the minimum space to write each of these values. The output produced by the statement of column A is

```
1 +
10
100
---
111
```

Since the statements of column B have width specifiers, each numeric field is right justified within the space indicated by the specifiers. The output of the statements of column B is

```
  1 +
 10
100
---
111
```

EXAMPLE 3.15 What is the output of the following printf() statement?

```
printf("\n$3d%6.2f",1,47.9);
```

The output of this statement looks like this:

```
  1 47.90
```

The field width specifier 6.2 indicates that its associated number needs to be printed in a field that is 6 columns wide, with two decimal places. Since the number has only one decimal, the output is padded with an extra zero.

If the width specifier is smaller than the number that we want to print, the printf() function ignores the width specifier and uses the minimum number of spaces to print the number. Fractional parts of floating or double numeric values are always printed with the specified number of digits. As Example 3.15 illustrated, whenever the fractional part has fewer digits than specified, the number is padded with extra zeros. If the fractional part has more digits than specified, the number is rounded to the number of decimal places specified in the width specifier.

The scanf() function is the input counterpart of printf(). This function reads characters from the standard input, and interprets them according to the format indicated in the control string. The function stops reading, provided that there are no errors, when it exhausts all its control characters. The variable list of this function indicates the location where the input should be stored. Each of the elements making up this list should be an address.

EXAMPLE 3.16 What is the result of executing the input operation shown below? Assume that the variables numl and num2 have been declared as integer variables.

```
scanf("%d%d", &numl, &num2);
```

'Ihis statement allows a user to input two integer values into the integer variables numl and num2. The first control character "%d" is matched with the address of the variable numl. Likewise, the second control character "%d" is matched with the address of the variable num2. Notice the use of the operator & to indicate the address of the variables. If we assume that the user types the values 25 and 35 and then hits the Enter key, the value of 25 will be stored into variable numl and the value 35 will be stored into the variable num2.

The functions scanf() and printf() are generally used together to elicit and capture the user's input. Example 3.17 illustrates this.

EXAMPLE 3.17 Write the necessary instructions to prompt the user to enter two integer values.

To accomplish this task, we need two pairs of instructions. Each pair consists of one printf() to prompt the user and one scanf() to capture the input values. These two instructions may look like this:

```
printf("\nPlease enter the first integer value =>");
scanf("%d",&num1);
printf("\nPlease enter the second integer value =>");
scanf("%d",&num1);
```

Notice that the first printf statement, after skipping to a new line, displays on the video screen something like this:

Please enter the first integer value =>

where we have indicated with the symbol, ■ , the position of the cursor after the first print statement has been executed. Notice that, when the user starts typing the integer number, the typing starts at the position indicated by the cursor. When the user presses the Enter key, whatever value is typed is then stored into the variable num1.

The second pair of instructions behaves exactly the same as the first pair.

EXAMPLE 3.18 Explain the result of executing the following instructions. Assume that the variables have been properly declared and match the control characters indicated in the control string.

```
printf('\n$s\n%s', "Input three values", "An integer, a
  single character and afloat:');
scanf('%d 8c 8f", &ivar, &cvar, &fvar); printf('\nYou typed
  the following: %5d , $c , %7.3f : , ivar, cvar, fvar);
```

Notice the use of the control character "%s" to display the prompts to the user. These two prompts are written in two separate lines. Observe how the escape sequence is being used to obtain this effect. Assuming that the user typed 123 A 3.5, the result of executing these instructions is

Input three values

An integer, a single character and a float: 123 A 3.5

You typed the following: 123, A, 3.500

Notice that, within the control string of the printf, the spaces after each control character are displayed in the output line "as is."

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.6

3.6 NHẬP XUẤT ĐƠN GIẢN

Hãy nhớ lại rằng quy trình căn bản của việc xử lý máy tính đó là nhập, xử lý và xuất. Các chương trình có thể đọc dữ liệu nhập và viết kết quả xuất vào file hoặc đĩa. Việc xử lý nhập xuất file sẽ được trình bày trong chương 7. Để sử dụng cho mục đích của các chương kế tiếp chúng ta giả sử rằng có một người dùng đang ngồi trên máy tính cung cấp dữ liệu từ bàn phím và muốn xem kết quả xuất trên màn hình. Từ bản thân chương trình này hỗ trợ cho quy trình xử lý thông tin mà người dùng đã cung cấp rồi hiển thị kết quả xuất. Các câu lệnh của chương trình phải hoàn tất tác vụ này.

Một trong số các sự khác biệt lớn nhất giữa các ngôn ngữ máy đó là cách mà chúng xử lý dữ liệu nhập và xuất, do đó mỗi một ngôn ngữ sẽ được điều phối một cách riêng biệt. Cũng vậy bên trong mỗi ngôn ngữ cũng có thể có nhiều phương pháp để nhập và xuất. Phương pháp đơn giản nhất sẽ được trình bày ở đây.

3.6.1 Truy cập C++ I/O

C++ không có bất cứ lệnh nhập xuất nào được tạo sẵn. Tất cả các trình biên soạn C++. Tuy nhiên, tất cả các trình C++ có chứa một gói các lớp hướng đối tượng được gọi là các lớp istream. Bộ biên soạn xem xét mọi thứ nhập vào và xuất ra khỏi các dòng dữ liệu. Chương trình này phải trích xuất thông tin đối tượng dòng nhập và chèn dữ liệu vào đối tượng dòng xuất. Các dòng được xem như là các đối tượng tách rời và các gói được gọi là một thư viện của các hàm để giúp nhà lập trình trong vấn đề xuất (istream) và vấn đề chèn (ostream), nhà lập trình phải chỉ định rằng các hàm này sẽ được dùng bằng cách cung cấp cho bộ biên soạn một thông điệp để đưa vào trong thư viện cần thiết. Cú pháp phải đưa vào thư viện này là.

```
#include <istream.h>
```

Lệnh phải được cung cấp trước tiên trước bất kỳ câu lệnh chương trình nào khác, và dấu # phải được đặt ở cột đầu tiên của dòng, bộ biên soạn sau đó phải tìm file tiêu đề (.h) giải thích tất cả các nguồn có thể được dùng trong chương trình, một số ví dụ được liệt kê dưới đây sẽ giải thích ở bảng 3-5.

Hàm dành cho nhập là cin (đọc là "see in" dành cho ConsoleInput) toán tử trích xuất có sẵn để sử dụng với bất kỳ kiểu dữ liệu được tạo sẵn nào. Bất cứ nội dung nào được gõ nhập tại bàn phím sẽ được xem xét để bảo đảm rằng nó là kiểu dữ liệu đúng, sau đó nó được đặt vào trong một biết chỉ định. Biến ký tự sẽ nhận giá trị ASCII bất kỳ nội dung nào được gõ nhập. Giá trị số sẽ chỉ chấp nhận một con số. Nếu

có bất cứ nội dung nào khác nào được gõ nhập thì chương trình sẽ bị hỏng và một chương trình sẽ bị ngắt với một thông báo lỗi về thời gian chạy của kiểu dữ liệu không tương kết.

***Bảng 3-5** Một vài nguồn thư viện iostream.h*

Resource	Type	Explanation
cin	object	Controls extraction from standard stream of bytes coming from keyboard
>>	extraction operator	Gets bytes from an input stream object
cout	object	Controls insertion to the standard stream of bytes going out to the screen
<<	insertion operator	Inserts bytes into the standard output stream object
cerr	object	Controls immediate insertion of messages to the standard stream of bytes going out to the screen
endl or '\n'	constant manipulator	Inserts newline character into the output stream object – makes the cursor go to the next line

***VÍ DỤ 3.5** Ví dụ sau đây minh họa việc sử dụng cin*

General Syntax: `cin>>variable;`

Examples:

```
cin>>num1;          // if a number is typed its value is
                       placed into num1
cin>>char2;         // whatever is typed its ASCII value
                       is placed into chart
cin>>num1>>num2;    // two numbers must be typed, sepa-
                       rated by a space
```


Hàm để xuất là cout (đọc là "see out" dành cho ConsoleOUTput). Một lệnh khác thường được dùng cho các thông báo lỗi đó là cerr (đọc là "see err" dành cho ConsoleError message) thông điệp này cũng được hiển thị trên màn hình, tuy nhiên nó trợ giúp cho nhà lập trình để phân biệt kết quả xuất bình thường với các thông báo lỗi đặc biệt mà chương trình đưa ra khi nó thực thi. Toán tử chèn có thể được chèn với bất cứ dữ liệu được tạo sẵn nào. Nếu biến là một giá trị số thì trước tiên nó sẽ được đổi thành cách trình bày ASCII để có thể được hiển thị trên màn hình. Bất cứ số các hạng mục nào cũng có thể được chèn vào giá trị dòng xuất bao gồm hằng số endl với cursor dòng kế tiếp. Chuỗi các ký tự được đặt giữa các dấu móc kép cũng có thể được hiển thị.

***VÍ DỤ 3.6** Phần sau đây minh họa cách sử dụng cout*

Statements where num1 is 4 and char1 is 'p':	**Output:**
`cout<<num1 ;`	4
`cout<<"The number is "<<num1<<endl;`	The number is 4
`cout<<"Your initial is "<<char1<<endl;`	Your initial is p

```
cout<<"line 1"<<endl<<"line 2"<<endl;        line 1
                                             line 2
cout<<"The sum of 4 and 5 is "<<4+5<<endl;   The sum of 4 and 5 is 9
```

Người lập trình cần phải kết hợp xuất và nhập để cung cấp một thông điệp cho người dùng về những gì đã được gõ nhập trên bàn phím. Không cần thiết người dùng phải biết kiểu nhập đó. Ví dụ các dòng C++ cần phải yêu cầu người dùng cần phải gõ nhập số nguyên dương rồi đọc nó thành một biến posInt sẽ là:

```
cout<< "Enter a number greater than zero =>";
cin>>posInt;
```

Khái niệm về việc đưa ra lời nhắc cho người dùng vô cùng quan trọng cho bất cứ ngôn ngữ nào. Các chỉ dẫn phải rõ ràng và chuyên biệt để thu nhỏ tối thiểu các lỗi trong thời gian chạy về kiểu không tương kết. Trong thực tế thì chương trình luôn luôn kiểm tra dữ liệu nhập cho phép đưa ra thông điệp có liên quan đến dữ liệu không đúng.

3.6.2 Java I/O

Như đã chỉ định ở phần đầu của chương, một chương trình Java có thể là một trình ứng dụng đứng riêng hoặc một applet được trộn vào trong một trang web. Quy trình nhập và xuất được xử lý rất khác nhau đối với hai tình huống này. Phần này chỉ trình bày quy trình nhập và xuất đơn giản nhất cần cho một trình ứng dụng Java bình thường, thư viện Java I/O phải được đưa vào hoặc được nhập để hoàn thành tác vụ này. Cú pháp dành cho tác vụ được minh họa dưới đây. Dấu asterisk (ký tự thay thế) có nghĩa rằng tất cả các hàm và đối tượng đều có sẵn trong thư viện Java I/O sẽ được nhập.

```
import java.io.*;
```

Java sử dụng đối tượng ***System*** *để đọc và in đối tượng dòng. Bảng 3-6 trình bày các hàm I/O phổ biến nhất được dùng trong Java.*

Bảng 3-6 *Các lệnh Java I/O*

Command	Explanation
System.out.print(..)	Insertion of whatever is in the parentheses into the standard stream of bytes going out to the screen
System.out.println(..)	Insertion of whatever is in the parentheses into the standard stream of bytes going out to the screen, then inserts newline character
System.out.flush()	Forces whatever is in the buffer to be displayed before reading
System.in.read()	Controls extraction from standard stream of bytes coming from keyboard
System.in.skip(n)	Skips n characters of input stream – used to pass unwanted values such as Enter

Cũng giống như C++, Java sử dụng một đối tượng để đọc các dòng dữ liệu được đưa vào từ bàn phím, System.in và một đối tượng khác để in

các dòng dữ liệu xuất sang màn hình. System.out Java (và C++ cũng thế) có một vị trí bộ nhớ tạm thời để giữ các dữ liệu nhập và xuất được gọi là bộ nhớ đệm (buffer). Nếu chương trình này có kết quả xuất đã được cho thì bộ nhớ đệm sẽ được làm sạch trước khi thử nhập dữ liệu nhập. System.out.flush() là một hàm có chức năng bảo đảm bộ nhớ đệm làm sạch.

Hệ thống nhập Java System giữ một ký tự một lần và giả sử rằng đây là một ký tự thậm chí nếu chúng ta đang tìm các số nguyên. Do đó chúng ta phải thay đổi một cách đặc biệt ký tự nhập sang một số nguyên. Hãy nhớ rằng ASCII của 0 là 48 của 1 là 49 v.v... do đó chúng ta có thể trừ giá trị ASCII của 0 hoặc 48 khỏi ký tự để thay nó sang một số nguyên. Ví dụ 3.7 sẽ in 5 nếu 5 được nhập vào. Nếu không có phép trừ, chúng ta phải in 53 hoặc giá trị ASCII của 5.

***VÍ DỤ 3.7** Phần minh họa sau đây sử dụng dữ liệu nhập và xuất trong hệ thống Java*

```
System.out.print ("Enter a number");
                                //give instructions to user
System.out.flush(); // flush the buffer before reading
num1=System.in.read() ;         // read the integer that is typed
num1=num1 - '0';                // subtract 48 to give the integer
                                // value, not ASCII
System.out.println ("Your number is"+num1);
```


Cũng cần lưu ý rằng system.in.read() sẽ chỉ chấp nhận một ký tự. Nếu 999 được nhập vào trong ví dụ trên đây, thì chỉ có số 9 đầu tiên là được đọc, nếu hai số hạng được đọc và nếu phím enter được nhấn giữa chúng, thì ký tự phải được lướt. Số các ký tự phải lướt thì phụ thuộc vào tình hình thực thi trong thực tế. Điều này được minh họa rõ trong ví dụ 3.8. input và output thực hiện dễ dàng với các trình ứng dụng nhỏ Java, điều này vượt qua khuôn khổ của cuốn sách này.

***VÍ DỤ 3.8** Phần sau đây minh họa thêm về các sự kiện nhập và xuất trong hệ thống Java*

```
System. out. print ("Enter a number");
System.out.flush();
num1=System. in. read () ;  //user types integer
                            // and then presses Enter
num1=num1-'0';
System. out. print ln ("Your number is"+num1);
System. in. skip (2) ;              //skip the Enter key before
                            //reading new data
System. out. print ("Enter a number");
```

```
System.out.flush();
num1=System.in.read();
num1=num1-'0';
System. out. println ("Your number is"+num1);
```

3.6.3 Visual Basic I/O

Visual Basic I/O không phức tạp như Java và C++. Dữ liệu nhập và xuất không phải là các dòng dữ liệu ký tự nhưng các giá trị rời rạc đặc biệt đến và ra. có nhiều cách để quản lý I/O trong một môi trường visual. Môi trường hướng đối tượng của VB sẽ được thảo luận chi tiết trong chương 8. trong chương này chỉ có I/O đơn giản nhất từ module mã mới giải thích.

Phần Visual của Visual Basic ngụ ý rằng có thể xem thấy các đối tượng đặc biệt trên màn hình. Quy trình nhập được hoàn thành thông qua việc sử dụng input boxes và quy trình xuất thông qua các hộp Message. Cú pháp dành cho các đối tượng này được minh họa trong bảng 3.7. Ở đây cũng mô tả lệnh xuất khác đó là Print. Lệnh này sẽ được dùng trong nhiều ví dụ VB trong sách này. Trong môi trường VB nó sẽ in kết quả xuất sang dạng chương trình chính hoặc cửa sổ.

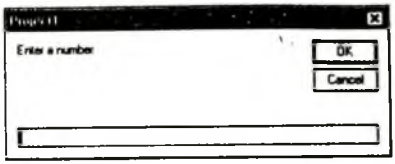
***Bảng 3-7** Visual Basic I/O*

VB Command Syntax	Explanation
variable = InputBox(*prompt*)	*prompt* is message to user, and *variable* is filled with value typed into box by user
MsgBox(*message*)	*message* is displayed in a box on screen
Print *variable or constant list*	Simple print used to print to VB form
Str(value)	Converts numbers into Strings that can be displayed in boxes – usually not necessary with Print
+ or &	Used to connect string parts into a long string to be displayed in boxes or with Print

Các ô input là các hộp thoại với các dòng ngắt được hiển thị và có một chỗ để người dùng trả lời, người dùng gõ nhập một giá trị vào trong ô, giá trị đó được đặt thành một biến. VB rất sinh động. Đối tượng nhập đi vào chương trình dưới dạng chuỗi. Tuy nhiên, nếu đích của biến là một số nguyên hoặc là một giá trị chính xác, thì đối tượng nhập được đổi một cách tự động thành kiểu dữ liệu đúng. Ví dụ 3.9 trình bày một số ví dụ hai lệnh Inputboxes với đối tượng xuất kết quả.

VÍ DỤ 3.9

```
num1=InputBox("Enter a number")
```

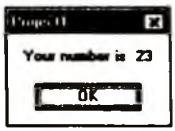



```
char1=InputBox("Enter a character")
```

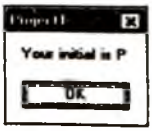
Các hộp Message hiển thị thông điệp trong một hộp thoại rồi chờ đợi người dùng nhấp OK. Có các tùy chọn dành cho các hộp Input và Message để cho phép hiển thị các nút khác nhau. Tuy nhiên hiện tại chúng ta sẽ sử dụng các giá trị mặc định. Thông điệp được gửi đến hộp Message phải là một chuỗi, nếu các số nguyên hoặc các số chấm động được hiển thị thì chúng phải được biến đổi sang một chuỗi với hàm Str(value). Tất cả các phần của thông điệp String đều được ghép nối với một dấu cộng hoặc một dấu ampersand. Ví dụ 3.10 trình bày một số ví dụ của lệnh Message box với đối tượng xuất kết quả. Lưu ý công dụng của ký tự ASCII 10 để cưỡng bức cursor sang dòng kế tiếp. Điều này tương ứng với endl của C++.

VÍ DỤ 3.10

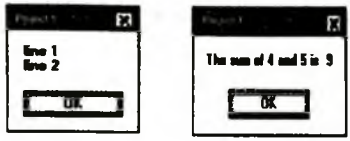
```
MsgBox ("Your number is"+Str(num1))
```




```
MsgBox ("Your initial is"+char1)
```



```
MsgBox ("line 1" & Chr(10) & "line 2")
MsgBox ("The sum of 4 and 5 is" & Str(4+5))
```



3.6.4 C I/O

Các hàm thư viện được dùng phổ biến nhất để thực hiện các phép tính I/O trong C đó là hàm printf() và scanf(). Hàm printf() cho phép chúng ta viết sang thiết bị xuất chuẩn, sang màn hình video, sang dữ liệu được định dạng và không được định dạng.

Cũng vậy hàm scanf() cho phép chúng ta đọc các giá trị từ thiết bị nhập chuẩn, bàn phím. Trong chương 7 chúng ta sẽ sử dụng các lệnh tương tự để thực hiện file I/O. trong chương này chúng ta sẽ minh họa cách sử dụng các hàm này với một vài ví dụ. Dạng tổng quát của hai hàm này như sau:

***printf**(control string, list of variables)*

***scanf**(control string, list of variables)*

Ở đây chuỗi control là một chuỗi các ký tự được đặt trong các dấu móc kép có thể chứa các đặc trưng điều khiển. Mặc dù mục đích của chuỗi điều khiển rất giống nhau ứng với các hai hàm nhưng vẫn có một sự khác biệt nhỏ trong công dụng của chúng. chuỗi control trong printf() thường chỉ cho ta biết cách dữ liệu được định dạng dành cho đối tượng xuất, có nghĩa rằng nó liên quan đến hình dáng của dữ liệu lúc được in ra. còn chuỗi control trong hàm scanf() là một trình tự điều khiển phép biến đổi có nghĩa rằng nó chỉ định dữ liệu đang được người đùng nhập vào. Danh sách của các biến trong printf() tương ứng với các biến hoặc các hằng dang được in, các biến và các hằng này được tách rời nhau bằng một dấu phẩy. Danh sách các biến trong scanf() tương ứng với danh sách các biến mà giá trị của nó phải được đọc. Tuy nhiên, không giống như sprintf(), hàm scanf() yêu cầu rằng địa chỉ của các tuyến thay vì tên của chúng, phải được dùng trong danh sách của các biến. Các biến trong danh sách này cũng phải được tách rời nhau bởi một dấy phẩy. Tổng quát này đối với hai hàm printf() và scanf() cần phải có mối quan hệ một đối một giữa các ký tự biến đổi và các biến hoặc các hằng trong danh sách biến. Để sử dụng bất cứ hàm nào trong số hàm này chúng ta cần phải đưa vào file tiêu đề stdio.h

Tất cả các đặc trưng biến đổi bắt đầu bằng ký tự phần trăm và kết thúc bằng một ký tự biến đổi. Danh sách từng phần của các ký tự biến đổi thường được dùng phổ biến nhất liên quan đến printf() và scanf() được cho trong bảng 3-8 và 3-9 tương ứng.

Lúc thông tin xuất sang một thiết bị giống như màn hình video bên cạnh các ký tự biến đổi chuỗi điều khiển cần phải có chứa các dãy trình tự thoát. Một dãy trình tự thoát chính là một ký tự đơn được trình bày bởi một tổ hợp của hai ký tự khác. ký tự đầu tiên trong số hai ký tự này chính là ký tự backslash "\"; ký tự thứ hai của chuỗi trình tự thoát là một mẫu tự. Bảng 3-10 minh họa danh sách từng phần của các chuỗi trình tự thoát.

Bảng 3-8

Conversion Character for printf()	Associated Argument is Printed
c	as character
d	as decimal
f	as floating point
s	as a string

Bảng 3-9

Conversion Character for scanf()	Associated Argument is Converted to
c	a character
d	a decimal
f	a floating-point number
s	a string
Lf	a double

Bảng 3-10

Escape Sequence	Effect
\a	make a beep sound
\b	backspace
\f	new page or form
\n	new line
\t	horizontal tab

VÍ DỤ 3.11

Hãy viết thông điệp "Hello World" vào màn hình. Để viết thông điệp vào màn hình chúng ta có thể sử dụng hàm printf() ở dạng đơn giản nhất. Để in thông điệp đã cho chúng ta có thể sử dụng hàm frintf() như sau:

```
printf("Hello World")
```

Câu lệnh này sẽ in Hello World trên màn hình bắt đầu ở vị trí hiện tại của cursor có nghĩa rằng nếu chúng ta giả sử cursor đang ở cột 10 trên màn hình, thì lệnh này sẽ in thông điệp bắt đầu trên cột 10. Tuy nhiên, nếu chúng ta cần in thông điệp này trên cột 1 ở dòng kế tiếp thì lệnh cần phải được chỉnh sửa như sau:

```
printf("\Hello World");
```

Lưu ý rằng trong trường hợp này chúng ta đã sử dụng chuỗi trình tự thoát "\n" như được chỉ định trong bảng 3-10, ảnh hưởng của chuỗi trình tự này đó là "nó đi vào một dòng mới" trước khi in thông điệp.

VÍ DỤ 3.12 *Hãy viết thông điệp "I am fine" bắt đầu trên một dòng mới, có 5 khoảng trống nằm phía bên phải. Hãy gợi sự chú ý của người dùng bằng cách tạo nên một âm thanh beep.*

Để thực hiện tất cả điều này hàm printf() cần phải được viết như sau:

```
printf("\n\tI am fine\a");
```


Trong câu lệnh này ảnh hưởng của chuỗi trình tự thoát đi từ trái sang phải như sau:

"\n" di chuyển con trỏ sang một dòng mới.

"\t" di chuyển con trỏ đến điểm tab kế tiếp. Các điểm tab này cách nhau 5 khoảng trắng

Như được sử dụng trước đây hàm printf() cũng có thể được dùng để kiểm soát diện mạo của đối tượng xuất. Điều này thường được hoàn tất bằng cách sử dụng một tổ hợp các ký tự chuỗi điều khiển và những số chỉ định chiều rộng trường. Một số chỉ định chính là một con số nguyên được viết giữa dấu % và mẫu tự điều khiển, để chỉ định cùng với nội dung khác số các cột xuất có sẵn để viết một biến hoặc một hằng. Lúc số chỉ định được dùng các giá trị số được canh phải và các chuỗi ký tự được canh trái. Ví dụ sau đây minh họa điều này.

VÍ DỤ 3.13 *Tìm kết quả xuất của câu lệnh printf() sau đây?*

```
printf ("The sum of %5d and %5d is %7d" , 20, 25, 45) ;
```

Bên trong chuỗi điều khiển, có ba ký tự điều khiển, mỗi trong số ba ký tự này được liên kết với 1 và chỉ 1 trong số các hằng xuất hiện trong "danh sách biến". Sự liên kết xảy ra từ trái sang phải, giữa ký tự điều khiển, số chỉ định chiều rộng và các hằng số được cho như sau:

%5d được liên kết với 20	Viết số nguyên 20 bằng cách sử dụng trường có chiều rộng 5 cột
%5d được liên kết với 25	Viết số nguyên 25 bằng cách sử dụng trường có chiều rộng 5 cột
%7d được liên kết với 45	Viết số nguyên 45 bằng cách sử dụng trường có chiều rộng 7 cột

Do đó kết quả xuất sẽ giống như dưới đây:

The sum of 20 and 25 is 45

Lưu ý rằng tất cả các giá trị số đều được canh phải bên trong các trường của chúng. điều này cho thấy rằng các số "được viết ngược" có nghĩa rằng chữ số cuối cùng của con số được viết trong cột cuối cùng của trường; các chữ số còn lại được viết "di chuyển hướng về phía trái". Hàm này viết số 20 trong một trường có chiều rộng là 5 cột do đó có 3 khoảng trống nằm phía bên trái của chữ số 2. tình huống tương tự cũng xảy ra với các hằng số khác.

VÍ DỤ 3.14 *Có sự khác biệt nào giữa kết quả xuất do các câu lệnh của cột A tạo ra và các câu lệnh print của cột B tạo ra?*

Giả sử rằng các câu lệnh này trong mỗi cột được thực thi theo một thứ tự được chỉ định và độc lập với các câu lệnh cột khác.

Column A	Column B
`printf("\n%d+",1);`	`printf("\n%5d+",1);`
`printf("\n%d",10);`	`printf("\n%5d",10);`
`printf("\n%d",100);`	`printf("\n%5d",100);`
`printf("\n_____");`	`printf("\n_____");`
`printf("\n%d",111);`	`printf("\n%5d",111);`

Lưu ý rằng các câu lệnh trong cột A không có một số chỉ định chiều rộng. Trong trường hợp này khi C sử dụng khoảng trống tối thiểu để viết mỗi trong số các giá trị này. Kết quả xuất được tạo ra bởi câu lệnh là cột A.

```
1 +
10
100
-----
111
```

Bởi vì các câu lệnh là cột B có các số chỉ định chiều rộng cho nên mỗi một trường số phải được canh chỉnh bên phải với khoảng trống do các số chỉ định xác lập. Kết quả xuất có các câu lệnh ở cột B là

```
  1 +
 10
100
-----
111
```

VÍ DỤ 3.15 *Tìm kết quả xuất của câu lệnh printf() sau đây*

```
printf("\n$3d %6.2f",1,47.9);
```

Kết quả xuất của câu lệnh này giống như dưới đây:

1 47.90

Số chỉ định chiều rộng của trường là 6.2 chỉ định rằng con số liên kết của nó cần phải được in trong một trường có độ rộng 6 cột, với 2 chữ số thập phân. Bởi vì số này chỉ có một chữ số thập phân nên kết quả xuất được đính kèm bằng một số 0 dư.

Nếu số chỉ định chiều rộng nhỏ hơn số mà bạn muốn in, thì hàm printf() bỏ qua số chỉ định chiều rộng và sử dụng số cực tiểu các khoảng trống để in con số. Phần phân số của dấu chấm động hoặc các giá trị số đôi luôn luôn được in với số các chữ số được chỉ định. Như ví dụ 3.15 minh họa, bất kỳ lúc nào phần phân số có chữ số ít hơn chữ số đã được chỉ định, con số được đính thì con số phải được các số 0 đính vào. Nếu phần phân số có nhiều chữ số hơn là số được chỉ định thì con số này được làm tròn sang số các chữ số thập phân được chỉ định trong số chỉ định chiều rộng. Hàm scanf() là phần nhập của printf() hàm này đọc các ký tự từ phần nhập chuẩn in chúng theo dạng chỉ định trong chuỗi điều khiển. Hàm này ngưng đọc, điều này cho thấy rằng không có lỗi nào lúc nó dùng hết tất cả các ký tự điều khiển. Danh sách biến này chỉ ra vị trí nơi mà đối tượng nhập được lưu trữ. Mỗi một phần tử làm nên danh sách này sẽ là một địa chỉ.

VÍ DỤ 3.16 *Tìm kết quả việc thực thi phép tính nhập được minh họa dưới đây? Giả sử rằng các biến num1 và num2 được khai báo dưới dạng các biến nguyên.*

```
scanf("%d%d", &num1, &num2);
```

Câu lệnh này cũng cho phép người dùng nhập vào hai giá trị nguyên vào các biến nguyên num1 và num2. ký tự điều khiển đầu tiên là "%d" được tương kết với địa chỉ của biến num1. cũng vậy ký tự điều khiển thứ 2 là "%d" được tương kết với địa chỉ của biến num2. lưu ý công dụng của toán tử & để chỉ ra địa chỉ của các biến. Nói chung ta giả sử rằng người dùng đã nhập các giá trị 25 và 35 và sau đó nhấn vào bàn phím thì giá trị 25 sẽ được lưu trữ thành biến num1 và giá trị 35 sẽ được lưu trữ thành biến num2.

Các hàm scanf() và printr() thường được dùng với nhau để làm rõ ra và bắt giữ đối tượng nhập của người dùng. Ví dụ 3.17 minh họa điều này.

VÍ DỤ 3.17 *Hãy viết các câu lệnh cần thiết để nhắc nhở người dùng phải nhập vào hai giá trị nguyên. Để hoàn tất tác vụ này chúng ta cần hai cặp lệnh. Mỗi cặp có chứa một printf() để nhắc người dùng và một có chứa scanf() để bắt giữ các giá trị nhập. Hai lệnh này giống như dưới đây:*

```
printf("\nPlease enter the first integer value =>");
scanf("%d",&num1);
printf("\nPlease enter the second integer value =>");
scanf("%d",&num1);
```

Lưu ý rằng lệnh printf() đầu tiên, sau khi lướt sang một dòng mới, hiển thị lên màn hình video một vài nội dung như dưới đây:

Please enter the first integer value => ■

Ở đây chúng ta đã chỉ định ký hiệu ■ vị trí của cursor sau khi câu lệnh in đầu tiên được thực thi. Lưu ý rằng lúc người dùng bắt đầu gõ nhập số nguyên thì việc gõ nhập khởi đầu tại vị trí mà cursor chỉ định. Lúc người dùng nhấn phím enter bất cứ giá trị nào được gõ nhập sẽ được lưu trữ thành biến num1.

Các lệnh thứ hai có đặc tính giống hệt như cặp lệnh đầu tiên.

VÍ DỤ 3.18 *Giải thích kết quả của việc thực thi các lệnh sau đây. Giả sử rằng các biến được khai báo hoàn chỉnh và tương kết với các giá trị điều khiển được chỉ định trong chuỗi điều khiển.*

```
printf('\n$s\n%s', "Input three values", "An integer, a
  single character and afloat:');
scanf('%d 8c 8f", &ivar, &cvar, &fvar); printf('\nYou
  typed the following: %5d , $c , %7.3f : , ivar, cvar,
  fvar);
```

Lưu ý công dụng của ký tự điều khiển "%s" để hiển thị các dòng nhắc trong nội dung. Hai dòng nhắc này được viết thành hai dòng riêng biệt. Hãy quan sát cách mà chuỗi trình tự thoát đang được dùng để tìm ảnh hưởng này. Giả sử rằng người dùng nhập 123 A 3.5 thì kết quả thực thi các lệnh này là

```
Input three values
An integer, a single character and a float: 123 A 3.5
You typed the following: 123, A, 3.500
```

Lưu ý rằng bên trong chuỗi điều khiển của hàm printf, các khoảng trống sau mỗi một ký tự điều khiển được hiển thị trên dòng xuất "as is".

CHỦ ĐIỂM 3.7

WRITING A COMPLETE PROGRAM
Viết một chương trình hoàn chỉnh

All that remains is to put everything - input, process, and output - into a whole program that the computer can understand. Again, each programming language does this in a different way. In each example, the program must have a main section of code to be executed. Chapter 5 deals with other functions and subroutines that can be used along with the main.

3.7.1 C and C++

The main function in C and C++ is enclosed with curly brackets {} to indicate where it starts and ends. The general order for the statements is, of course:

(1) declare variables
(2) get input
(3) do processing
(4) produce output

EXAMPLE 3.19 The short sample program in Fig. 3-2 uses this general outline as comments. Notice the compiler directive #include must precede the main section.

```
//First Program - gets a number, calculates and prints the square
#include <isostream.h>
void main()
{
   //declare variables
   int num, square;
   //1. Get the number
   cout<<endl<<'Enter the number you want to square=>';
   cin>>num;
   //2. Square it
   square=num*num;
   //3. Print out the square
   cout<<num<<'squared is'<<square<<endl;
}//end main
```

Fig. 3-2 First C++ program - square a number.

In the function heading void main( ), the empty parentheses indicate that there is nothing coming into the section, and the void shows there is nothing going out. Other functions which have values going in and/or out will be described in Chapter 5.

3.7.2 C

```
// First Program - gets a number, calculates and prints the square
#include <stdio.h>
void main()
{
int num, square; // declare variables
// Get the number
printf('\nEnter the integer number you want to square =>');
scanf('$d',&num);
//Square it square=num*num;
//Print the square
printf('\n%d squared is %d\n",num,square);
}
```

This program consists of only one function, the main function. As indicated in Chapter 5, this is a required function in any C program since it is the first function to be executed when the program starts running.

3.7.3 Java

Java also uses curly brackets to begin and end its main section. However, Java is fully object-oriented, which means it is class based and even the program itself is a class. Classes will be explained in Chapter 8. For the present, it is important to remember that the class name is also the application name.

EXAMPLE 3.20 Consider the program shown in Fig. 3-3. It is the Java version of the C++ program above.

```
//First Program - gets a number, calculates and prints the square

import java.io.*;

class SquareIt {
   public static void main (String args[]) throws IOException
   {
      //declare variables
      int num,square;

      //1. Get the number
      System.out.print ('Enter a number between 0 and 9 you want to square => ');
      System.out.flush();       //clear buffer before reading
      num=System.in.read();
      num=num-'0';              //change character input to integer

      //2. Square it
      square=num*num;

      //3. Print out the square
      System.out.println (num+' squared is ' +square);
   }//end main
}//end class
```

Fig. 3-3 First Java program - square a number.

There are two lines to examine. The first is:

class SquareIt {

This line indicates that the application is called Squareit. In Java the file must have the same name as the class. Everything in the program is within this class definition. Notice the bracket ending the class is the last line of the file.

The second line to examine is:

public static void main (String args[]) throws IOException

This line obviously begins the main function. It is required to be typed exactly like this. The void *main (String argso) indicates* it is the main function which can receive string messages from the operating system and is not returning a value. The *public* and *static* indicate that the program can be run from outside itself (which obviously is necessary when we start execution) and that it will remain in memory until it is finished. If errors occur in I/O, Java generates an exception to tell us what happened. The final part of the line, *throws IOException,* means that input and output will be performed and we know that an error might occur, but we will ignore these exceptions and let the program crash. Writing code to handle all exceptions is beyond the scope of this book.

3.7.4 Visual Basic

Visual Basic calls its functions Sub's, or subroutines. When writing code modules, instead of using the Visual tools, we are required to have a *Sub main.*

EXAMPLE 3.21 The Visual Basic version of the squaring program is found in Fig. 3-4.

```
'First Program - gets a number, calculates and prints the square

Sub main()
  ' declare variables
  Dim num As Integer, square As Integer

  '1. Get the number
  num=InputBox("Enter the number you want to square =>")

  '2. Square it
  square=num*num

  '3. Print out the square
  MsgBox (Str(num)+" squared is "+Str(square))

End Sub
```

Fig. 3-4 Visual Basic program -square a number

Notice the Sub must begin with its name, main, and the empty parentheses indicating nothing is coming into the program. When all the code has executed, the Sub must End. Everything else in the Sub is self-explanatory.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 3.7

3.7 HÃY VIẾT MỘT CHƯƠNG TRÌNH HOÀN CHỈNH

Tất cả những gì còn lại đó là đưa mọi thứ "đối tượng nhập, xử lý và xuất" – thành một chương trình hoàn chỉnh mà máy tính có thể hiểu được. Một lần nữa, ngôn ngữ lập trình thực hiện điều này theo một cách thức khác nhau. Trong mỗi ví dụ, chương trình phải có một phần mã main để thực thi. Chương 5 xử lý các hàm và các thủ tục con khác có thể được dùng cùng với phần main.

3.7.1 C và C++

Hàm main trong C và C++ được đặt bên trong các dấu ngoặc móc để chỉ ra nơi mà nó bắt đầu và kết thúc. Thứ tự chung của câu lệnh như sau:

(1) khai báo biến

(2) nhận đối tượng nhập

(3) xử lý

(4) tạo kết quả xuất

VÍ DỤ 5.19 *Chương trình mẫu trong hình 3-2 sử dụng khung bố cục tổng quát làm phần bình phẩm. Lưu ý dấu # phải được đặt trước phần chính.*

```
//First Program - gets a number, calculates and prints the square
#include <iostream.h>
void main()
{
   //declare variables
   int num, square;
   //1. Get the number
   cout<<endl<<"Enter the number you want to square=>";
   cin>>num;
   //2. Square it
   square=num*num;
   //3. Print out the square
   cout<<num<<"squared is"<<square<<endl;
}//end main
```

Hình 3-2 *Chương trình C++ đầu tiên – bình phương một số*

Trong tiêu đề hàm void main() phần dấu móc đơn trống chỉ ra rằng không có gì được đưa vào trong mục này. Chữ void cho thấy rằng không có gì xuất ra. các hàm khác có các giá trị vào và ra sẽ được mô tả trong chương 5.

3.7.2 C

```
// First Program - gets a number, calculates and prints the
  square
#include <stdio.h>
void main()
{
int num, square; // declare variables
// Get the number
printf('\nEnter the integer number you want to square =>');
scanf('$d',&num);
//Square it square=num*num;
//Print the square
printf('\n%d squared is %d\n",num,square);
}
```

Chương trình này chỉ có một hàm đó là hàm main. Như được chỉ định trong chương 5, đây là một hàm cần thiết trong bất kỳ chương trình thực thi nào, bởi vì nó là hàm đầu tiên phải được thực thi lúc chương trình bắt đầu chạy.

3.7.3 Java

Java cũng sử dụng các dấu móc ngoặc để bắt đầu và kết thúc mục chính. Tuy nhiên, Java là ngôn ngữ hướng đối tượng, có nghĩa là nó là một lớp cơ sở và thậm chí chương trình từ bản thân cũng là một lớp. Các lớp sẽ được giải thích trong chương 8. Còn hiện tại thì chúng ta cần nhớ rằng tên của lớp cũng là tên của trình ứng dụng.

***VÍ DỤ 3.20** Hãy khảo sát chương trình được minh họa trong hình 3-3 dưới đây là phiên bản Java của chương trình C++ trên đây.*

```
//First Program - gets a number, calculates and prints the square

import java.io.*;

class SquareIt {
   public static void main (String args[]) throws IOException
   {
      //declare variables
      int num,square;
```

***Hình 3-3** Chương trình Java đầu tiên – bình phương một số*

```
        //1. Get the number
        System.out.print ("Enter a number between 0 and 9 you want to square => ");
        System.out.flush();        //clear buffer before reading
        num=System.in.read();
        num=num - '0';             //change character input to integer

        //2. Square it
        square=num*num;

        //3. Print out the square
        System.out.println (num+" squared is " +square);
    }//end main
}//end class
```

***Hình 3-3** (tiếp theo)*

Có hai dòng cần xem xét. Trước tiên là

Class SquareIt {

Dòng này chỉ ra rằng trình ứng dụng được gọi là SquareIt, trong Java file này phải có tên giống hệt như tên trong lớp. Mọi thứ trong chương trình phải được nằm bên trong định nghĩa lớp này. Lưu ý dấu móc kết thúc của lớp là phải nằm trên dòng cuối cùng của file. Dòng thứ hai phải xem xét là

public static void main (String args[]) throws IOException

Dòng này hiển nhiên bắt đầu bằng hàm chính. Nó phải được gõ nhập một cách chính xác như thế. Void main (String args[]) chỉ ra đây là một hàm chính có thể nhận các thông điệp chuỗi từ hệ điều hành và không trả về một giá trị. publuc và static chỉ ra rằng chương trình có thể chạy từ bên ngoài nó (hiển nhiên điều này cần thiết lúc chúng ta bắt đầu thực thi) và rằng nó sẽ giữ lại trong bộ nhớ cho đến khi hoàn tất. Nếu xảy ra lỗi trong I/O thì Java tạo nên một ngoại lệ để báo cho chúng ta những gì xảy ra. phần cuối cùng của dòng I/O Exception, có nghĩa rằng đối tượng nhập và xuất sẽ được thực hiện và chúng ta biết có một lỗi sẽ xảy ra, nhưng chúng ta sẽ bỏ qua những ngoại lệ này và để cho chương trình hỏng. Việc viết các mã để xử lý các ngoại lệ vượt xa khuôn khổ của quyển sách này.

3.7.4 Visual Basic

Visual Basic gọi các hàm của nó là Sub hoặc thường trình con. Lúc viết các module của mã thay vì sử dụng các công cụ Visual chúng ta cần phải có Sub main.

***VÍ DỤ 3.21** Phiên bản Visual Basic của hàm bình phương được cho trong hình 3-4*

```
'First Program - gets a number, calculates and prints the square

Sub main()
  ' declare variables
  Dim num As Integer, square As Integer

  '1. Get the number
  num=InputBox('Enter the number you want to square =>')

  '2. Square it
  square=num*num

  '3. Print out the square
  MsgBox (Str(num)+" squared is "+Str(square))

End Sub
```

Hình 3-4 *Chương trình đầu tiên Visual Basic – bình phương một số*

Lưu ý Sub phải bắt đầu bằng tên của nó là main và các dấu móc ngoặc trống chỉ ra không có gì được đưa vào trong chương trình. Lúc tất cả mã được thực thi, Sub phải có End. Mọi thứ khác trong Sub có phần giải thích.

SOLVED PROBLEMS
Bài tập có lời giải

3.1 What would be the ASCII representation of the word "HELLO" in decimal? By looking at the ASCII chart in Table 3-1, the following values can be found:

H=>72
E=>69
L=>76
L=>76
O=>79

Therefore, the answer would be: 72 69 76 76 79

3.2 What does the following ASCII code represent?

69 97 116 32 118 101 103 103 105 101 115 33

Answer. Eat veggies! Notice the space and the exclamation point are both represented explicitly

3.3 What does the following ASCII code represent?

51 43 49 61 52

Answer: 3 + 1 = 4

Notice each digit is a character, not a numeric value. Therefore, real arithmetic cannot be performed. 51 plus 49 does not equal 52. This is simply the ASCII representation of the equation.

3.4 The **sizeof(variable)** displays the number of bytes used, and the & operator gives the actual address in memory. This address can be different on each computer, or even for each run of the program. What will be the output of the following C++ code section?

```
int int1=1;        short short1=1;    long long1=1;
double d1=1.0;     float f1=1.0;      char ch='1';
cout<<'int1\t'<<int1<<' '<<sizeof(int1)<<' '<<&int1<<endl;
cout<<'short1\t'<<short1<<' '<<sizeof(short1)<<' '<<&short1<<endl;
cout<<'long1\t'<<long1<<' '<<sizeof(long1)<<' '<<&long1<<endl;
cout <<'d1\t'<<d1<<' '<<sizeof(d1)<<' '<<&d1<<endl;
cout<<'f1\t'<<f1<<' '<<sizeof(f1)<<' '<<&f1<<endl;
```

```
int1    1 4 0x0066FDF4
short1  1 2 0x0066FDF0
long1   1 4 0x0066FDEC
d1      1 8 0x0066FDE4
f1      1 4 0x0066FDE0
```

Answer: Notice the addresses are represented by the computer in hexadecimal notation.

3.5 Write the statements needed to declare the variables and constants for a program to find the circumference and area of a rectangle. Show the declarations in Visual Basic as well as GC++ and Java.

Visual Basic	GC++	Java
`Const SIDES=4`	`const SIDES=4;`	`final SIDES=4;`
`Dim length As Integer`	`int length;`	`int length;`
`Dim width As Integer`	`int width;`	`int width;`
`Dim areaOfRect As Integer`	`int areaOfRect;`	`int areaOfRect;`
`Dim circum As Integer`	`int circum;`	`int circum;`

3.6 Write the statements needed to declare the variables and constants for a program to find the circumference and area of a circle. Show the declarations in Visual Basic as well as GC++ and Java.

Visual Basic	GC++	Java
`Const PI=3.14`	`const PI=3.14;`	`final PI=3.14;`
`Dim radius As Integer`	`int radius;`	`int radius;`
`Dim areaofCircle As Single`	`float areaofCircle;`	`float areaofCircle;`

3.7 Specify whether these variable names are legal and/or meaningful: (a) @amount; *(b)* z; (c) sidel; *(d)* const; (e) my_password.

Variable Name	Explanation
(a) @amount	**Illegal** because of @ character
(b) z	**Legal**, but should not be used because it is not meaningful
(c) sidel	**Legal** and meaningful
(d) const	**Illegal** because it is a reserved word
(e) my-password	**Legal** and meaningful

3.8 Specify whether these constant names are legal and/or meaningful: (a) AMT#MONEY; *(b)* X;

(c) int rate; *(d)* Boolean; (e) MINCOST.

Variable Name	Explanation
(a) AMT#MONEY	**Illegal** because of # character
(b) X	**Legal**, but should not be used because it is not meaningful
(c) int_rate	**Legal** and meaningful, but not correct because constants are usually in all caps
(d) Boolean	**Illegal** because it is a reserved word
(e) MINCOST	**Legal** and meaningful

3 9 Write a short C++ program implementing a simple Adder program that will add two numbers.

```
void main()
{

   //declare variables
   int num1, num2, sum;

   //get 2 numbers
   cout<<endl<<"Enter number one=>"; cin>>num1;
   cout<<endl<<"Enter number two=>"; cin>>num2;

   //add them together
   sum=num1+num2;

   //print the result
   cout<<"The sum of"<<num1<<" plus "<<num2<<" is "<<sum<<endl;

}
```


3.10 Write the VB program implementing a simple payroll program. Assume the person makes $10.00 per hour with no extra for overtime.

```
Sub main()
 'declare variables
 Const PERHOUR = 10
 Dim hours As Integer, pay As Integer

 '1. get the number of hours
 hours=InputBox("How many hours?")

 '2. calculate pay
 pay=hours*PERHOUR

 '3. print out pay amount
 MsgBox ("Your pay this week is "+Str(pay))
End Sub
```

3.11 Write a Java program which calculates the area of a small circle with radius between 0 and 9.

```
//Area: calculate the area of a circle with radius between 0 and 9

import java.io.*;

class Area {
 public static void main ( String args[] ) throws IOException
 {
    // declare variables
    final double PI=3.14;
    int radius;
    double area;

//1. Get the radius
System.out.print ("Enter a radius of a circle between 0 and 9=>");
System.out.flush();
radius=System.in.read();
radius=radius - '0'; //change character to integer

//2. Calculate area
area= PI * (radius*radius);

//3. Print out the square
System.out.println ("The area of a circle with radius "+radius+" is "+area);
}// end main
}// end class
```


3.12 In C, display on the video screen the words "It is fun to do ISO".

Since we have to display a message on the screen, we can use the printf() function. Therefore, we may write the following statement

```
printf("\nIt is fun to do I\O");
```

However, if we try to run a program with this statement in it, we will get a warning from the compiler that may read something like this: "warning C4129: 'O' : unrecognized character escape sequefice". What this warning means is that the compiler (see Appendix A) could not recognize \O as a valid escape sequence. That is, the compiler "thinks" that we are trying to use an escape sequence instead of writing a backward slash. To remedy this, we need to use a sequence of two consecutive backward slashes. The correct printf statement should look like this:

```
printf("\nIt is fun to do I\\O");
```

3.13 Using only one printf() statement display on the screen, on three separate lines, the message "I came, I saw, and I conquered".

To display this message on the screen we use the printf() function. To write the message on three separate lines, we use the escape sequence "\n". Therefore, the C statement to accomplish this may look like this,

```
printf("\nI came,\nI saw and,\n I conquered");
```

3.14 Write the necessary C statements to print the numbers 1 throu 10 and their squares. Align the output.

The code to accomplish this may look like this:

```
for (i=1;i<=10;i++)
 printf("%4d\t%6d\n",i,i*i);
```

Notice the use of the width specifiers to align the results. See pa 101 for explanation of for statement.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

3.1 Hãy biểu thị ASCII dưới từ "HELLO" cơ sở thập phân?

3.2 Mã ASCII sau đây biểu thị điều gì?

3.3 Mã ASCII sau đây biểu thị điều gì?

3.4 Sizeof(variable) hiển thị số các byte được dùng, số toán tử & cho chúng ta địa chỉ thực sự trong bộ nhớ. Địa chỉ này có thể khác nhau trên mỗi một máy tính thậm chí khác nhau với mỗi một chương trình chạy. Hãy tìm kết quả xuất của phần mã C++ sau đây.

3.5 Hãy viết các câu lệnh cần thiết để khai báo các biến và các hằng dành cho một chương trình qua đó tìm chu vi và diện tích của một hình chữ nhật. Hãy trình bày các phần khai báo trong Visual Basic cũng như trong C, C++ và Java.

3.6 Hãy viết các câu lệnh cần thiết để khai báo các biến và các hằng dành cho một chương trình để tìm chu vi và diện tích của một hình tròn. Hãy trình bày các khai báo trong Visual Basic cũng như trong C, C++ và Javc.

3.7 Hãy chỉ định cho biết các tên biến sau đây là hợp pháp và - hoặc có nghĩa hay không: (a)@amount; (b)z; (c)side 1; (d)const; (e)my_password.

3.8 Chỉ định cho biết các tên hằng sau đây là hợp lý và có nghĩa hay không: (a) AMT#MONEY; (b) X; (c)int_rate; (d) Boolean; (e)MINCOST

3.9 Hãy viết một chương trình ngắn C++ để thực thi một chương trình Adder đơn giản qua đó cộng hai số

3.10 Hãy viết chương trình VB thực thi một chương trình tính lương đơn giản. Giả sử người này làm công 10 đô la trên mỗi giờ và không có làm ngoài giờ.

3.11 Hãy viết một chương trình để tính diện tích của một hình tròn có bán kính nằm giữa 0 và 9.

3.12 Trong C hãy hiển thị trên màn hình video các từ "It is fun to do I/O"

3.13 Bằng cách chỉ sử dụng chỉ một câu lệnh printf() hãy hiển thị trên màn hình, trên ba dòng riêng biệt, thông điệp "I came I saw and I conquered".

3.14 Hãy viết các câu lệnh khi cần thiết in các số từ 1 đến 10 và bình phương của chúng. Hãy canh chỉnh kết quả xuất.

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

3.15 What would be the ASCII representation of the phrase "Happy Programming!" in decimal?

3.16 What does the following ASCII code represent?

84 111 109 32 104 97 115 32 36 51 50 46 52 53 46

3.17 What does the following ASCII code represent?

71 111 32 70 105 115 104 108 110 103 33

3.18 Write the statements needed to declare the variables and constants for a program to find the miles per gallon for a specific car. Show the declarations in Visual Basic as well as C++ and Java.

3.19 Write the statements needed to declare the variables and constants for a program to handle an ATM machine. The transactions are specified by "W" for withdrawal or "D" for deposit.

3.20 Which of these identifiers are legal and/or meaningful?

Variables	Constants
(a) 123	(e) pi
(b) length	(f) MAXSTUDENTS
(c) test2	(g) RATE!
(d) a	(h) tempNum

3.21 Write a C++ program to convert a Celsius temperature to Fahrenheit.

3.22 Write a VB program to find the average of three test scores

3.23 Write a Java program that calculates the quotient and remainder of 97 divided by some integer between 1 and 9.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

3.15 Cách trình bày ASCII của cụm từ "Happy Programming" dưới dạng cơ số 10 như thế nào?

3.16 Mã ASCII sau đây trình bày điều gì?

3.17 Mã ASCII sau đây trình bày điều gì?

3.18 Hãy viết các câu lệnh cần thiết để khai báo các biến và các hằng dành cho một chương trình. Qua đó tìm số dặm đi trên mỗi gallon nhiên liệu với một xe đặc biệt. Hãy trình bày các khai báo trong Visual Basic cũng như trong C, C++ và Java.

3.19 Hãy viết các câu lệnh cần thiết để khai báo các biến và các hằng dành cho một chương trình xử lý một máy ATM. Các khoản giao dịch được chỉ định bởi "W" ứng với khoản rút tiền và "D" ứng với khoản gửi tiền.

3.20 Số nào trong các số nhận dạng sau đây là hợp lý và có ý nghĩa,

3.21 Hãy viết một chương trình C++ để đổi độ Celsius sang độ Fahrenheit.

3.22 Hãy viết một chương trình VB để tìm điểm trung bình trong ba điểm thi.

3.23 Hãy viết một chương trình Java để tính thương số và số dư khi chia 97 cho một số nguyên nằm giữa 1 và 9.

ANSWERS TO SUPPLEMENTARY PROBLEMS
Trả lời các bài tập bổ sung

3.15 72 97 112 112 121 32 80 114 111 103 114 97 109 109 105 110 103 33

3.16 Tom has $32.45.

3.17 Go Fshing!

3.18

Visual Basic	**C++ and Java**
`Dim milesTraveled As Single`	`float milesTraveled;`
`Dim gallonsUsed As Single`	`float gallonsUsed;`
`Dim mpg As Single`	`float mpg;`

3.19

Visual Basic	**C++ and Java**
`Dim transactionType As String`	`char transactionType;`
`Dim amount As Single`	`float amount;`
`Dim balance As Single`	`float balance;`

3.20 (a) 123 => Illegal, because it is a numeric constant. a123 would be legal but not meaningful.

(b) length => Legal and meaningful.

(c) test2 => Legal and meaningful.

(d) a => Legal, but not meaningful. It would be hard to debug in the program.

(e) pi => Legal, but constants are usually printed in all caps.

(f) MAXSTUDENTS -> Legal and meaningful.

(g) RATE! => Illegal because of the !.

(h) tempNum -> Legal, if that is the desired name. However, if it represents a temporary number that

may change values during the program, it would not be a constant.

3.21

```
#include <iostream.h>

void main()
{
    // declare variables
    double fahrenheit, celsius;

    //1. Get the Celsius temperature
    cout<<endl<<'Enter the Celsius temperature=>'; cin>>celsius;

    //2. Calculate the fahrenheit
```

```
    fahrenheit = (celsius *(9.0/5))+32;

    //3. Print out the result
    cout<<celsius<<' degrees Celsius is'
       <<fahrenheit<<' degrees fahrenheit'<<endl;
}
```

3.22

```
'Average - finds the average of three test scores
Sub main()
 'declare variables
 Dim test1 As Integer, test2 As Integer, test3 As Integer
 Dim average As Single

 '1. Get the 3 scores
 test1=InputBox('Enter the first score=>')
 test2=InputBox('Enter the second score=>')
 test3=InputBox('Enter the third score=>')

 '2. Calculate the Average
 average=(test1+test2+test3)/3

 '3. Print out the square
 MsgBox ('The average of'+Str(test1)+','+Str(test2) _
   ' and '+Str(test3)+' is '+Str(average))

 'notice the underline character to indicate continuation
End Sub
```

3.23

```
/* Divide - Calculates the quotient and remainder of 97
   divided by any number between 1 and 9 */

import java.io.*;

class Divide {
 public static void main ( String args[] ) throws IOException
 {
  // declare variables
  int num, quotient, remainder;

  //1. Get the divisor
  System.out.print ('Enter a number between 1 and 9=>');
  System.out.flush();
  num=System.in.read();
  num=num - '0'; // change character to integer

  //2. Calculate quotient and remainder
  quotient=97 / num; // integer division
  remainder=97 % num; // modulo arithmetic
```

```
    //3. Print out the result
    System.out.println ("When 97 is divided by "+num+" the quotient is "
      +quotient+ "and the remainder is"+remainder);
  }// end main
}// end class
```

CHAPTER 4

Control Structures and Program Writing

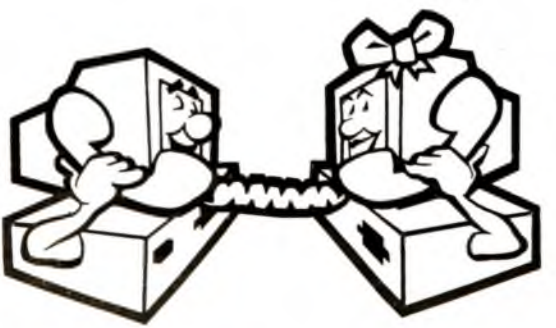
Các cấu trúc điều khiển và vấn đề viết chương trình

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các vấn đề về cấu trúc điều khiển và các tình huống gặp phải khi chương trình, cụ thể với các nội dung cơ bản sau đây:

- Boolean expressions
- Control structures - definitions
- Selection
- Repetition
- *Các biểu thức Boole*
- *Các cấu trúc điều khiển - các định nghĩa*
- *Sự lựa chọn*
- *Phép lặp*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.



CHỦ ĐIỂM 4.1

BOOLEAN EXPRESSIONS
Các biểu thức Boole

George Boole, a mathematician in the nineteenth century, was interested in the relationship between human logic and mathematics. He developed Boolean algebra, which is modeled after human thought patterns. This algebra is widely used today in the design of computer circuits, where all variables have only the values zero or one. A **Boolean expression** is a statement that is either true or false, one or the other, but not both at once. A Boolean expression may be a Boolean variable or constant by itself, or it may be constants or variables connected by relational or equality operators. Individual Boolean expressions can be combined into more complex statements by connecting them with the logical operators NOT, AND, and OR.

4.1.1 Boolean Variables and Constants

A Boolean expression may be a Boolean variable or constant. Such variables are often called flags because they can provide signals regarding the status of conditions in the program. It is important to give these variables descriptive names, because descriptive names help in the understanding and debugging of programs. Possible descriptive variable names include dataOK, workDone, or itemFound. Some languages have a built-in Boolean data type. Usually, however, integer variables are used to represent Boolean conditions. For example, in some languages zero (0) signifies false, and other non-zero values signify true. In C, C++, and Java, any value other than zero signifies true, but usually the value one (1) is used. In Visual Basic, negative one (-1) signifies true.

EXAMPLE 4.1 Demonstrate some Boolean variables in C. C++, and Java.

```
const TRUE=1;
const FALSE=G;
int dataOK, workDone, itemFound;
dataOK=FAL;-E; // set to false, data is NOT OK
workDone=T=-E;// set to true, the work is completed
itemFound-=-ELSE;// set to false, the item was not found
```

4.1.2 Relational Expression.

Boolean expressions may be construcred by connecting constants or variables by a relational operator. Relational operators and equality operators are shown in Table 4-1. Their English equivalents are used in normal speech and are easily understood.

Table 4-1 Relational Operators.

Relational Operators in C, C++ and JAVA	Relational Operators in Visual Basic	English Equivalent
<	<	less than
<=	<=	less than or equal to
>	>	greater than
>=	>=	greater than or equal to
Equality Operators	Equality Operators	
==	=	equal to
!=	<>	not equal to

Arithmetic calculations are always performed before the relational and equality operators are evaluated. For example, in the expression

```
4+5<20
```

the 4 and 5 would be added together before testing whether the result is in fact less than 20. All calculations and comparisons are performed left to right unless specified otherwise through the use of parentheses. The variables and/or constants connected by relational and equality operators must always be of the same type. For example, consider the values of the following expressions in Table 4-2. In the second column, both x and y have been declared type integer. In the third column both have been declared of type character.

Table 4-2 Examples of Boolean Expressions.

Boolean Expression	Value if x = 3 and y = 4	Value if x = 'b' and y = 'b'	Explanation
x > y	false	false	If x and y are the same, x is not greater than y
x == y	false	true	They are precisely equal to each other
x != y	true	false	They are precisely not equal to each other
x < y	true	false	If x and y are the same, x is not less than y
x <= y	true	true	If they are the same, then x is either less than or equal to y

EXAMPLE 4.2 What is the value of each of these expressions, given the values of x and y?

Boolean Expression	Value if x = 3 and y = 4	Explanation
x − 4 < y	true	(−1) is less than 4
x + 1 == y	true	4 is precisely equal to 4
x != y − 1	false	It is false that 3 is precisely not equal to 3
x + 1 >= y	true	3 is greater than or equal to 3

EXAMPLE 4.3 String variables can also be compared. In this case, the computer looks at the values left to right, character by character, and compares the ASCII value of each letter. Consider the Boolean expressions in table 4-3.

Table 4-3

Boolean Expression	Value	Explanation
"Ann" < "Annette"	true	The 4th character is considered to be null, which in ASCII is zero, and less than the ASCII value of any other character
"Sam" == "Sam"	true	Each character is precisely the same
"Johnson" <= "Johnsen"	false	The 6th character 'e' is less than 'o', which makes the entire string "Johnsen" less than the entire string "Johnson"

4.1.3 Logical Expressions

Individual Boolean variables or expressions can be combined into more complex statements by connecting them with logical operators. **Logical operators** for several languages are shown in the order of precedence in Table 4-4 with English explanations.

Table 4-4 Logical Operators.

Logical Operators in C, C++, and Java	English and Visual Basic	Explanation	Associativity
!	NOT	has single operand, and returns its negation (opposite)	Left to right
&&	AND	returns true ONLY if BOTH operands are true, false if EITHER or BOTH operands are false	Left to right
||	OR	returns true if EITHER or BOTH operands are true, false ONLY if BOTH operands are false	Left to right

The basic process for evaluating complex Boolean expressions is:

(1) Evaluate all the individual relational expressions left to right.
(2) Evaluate the logical expressions in parentheses.
(3) Evaluate the logical expressions in the order of precedence.

Table 4-5 demonstrates the order of precedence for all operators, including the arithmetic operators explained in Chapter 3. Each part of a Boolean expression is evaluated in the order indicated.

Table 4-5 Order of Precedence.

Operators	Operation	Order of Precedence	Associativity
( )	Parentheses	1	
^ (VB only)	Exponentiation	2	
! or NOT	Negation	3	Left to right
* /	Multiplication and division	4	Left to right
\(VB only)	Integer division	5	Left to right
% or MOD	Modulus	6	Left to right
+ −	Addition and subtraction	7	Left to right
< <= > >=	Relational operators	8	Left to right
== != or <>	Equality operators	9	Left to right
&& or AND	Logical AND	10	Left to right
|| or OR	Logical OR	11	Left to right

EXAMPLE 4.4 What is the value of each of these expressions given the values of x and y? Note, each individual Boolean expression is evaluated first, depending upon parentheses. Then the resultant Boolean combinations are evaluated next according to the precedence chart, and finally the whole expression is evaluated.

(*a*) if x = −3 and y = 4
(x == −3) && (y != 3)

```
  T      |    |     Evaluate this expression first, x is −3.
         |    T     Evaluate this expression second, y does not equal 3.
         T          Finally follow the rules for AND, true if both are true.
```

(*b*) if x = 3 and y = 3
(x == −3) && (y != 3)

```
  F      |    |     Evaluate this expression first.
         |    F     Evaluate this expression second.
         F          Finally follow the rules for AND, false if both are false.
```

EXAMPLE 4.5 What is the value of each of these expressions given the values of x and y? Note, each individual Boolean expression is evaluated first, depending upon parentheses. Then the resultant Boolean combina-

tions are evaluated next according to the precedence chart, and finally the whole expression is evaluated.

(*a*) if x = −3 and y = 4

```
(x > 0) && (y == 4)
  F     |       |
        |       T
        F
```

Evaluate this expression first.
Evaluate this expression second.
Finally follow the rules for AND, false if one side is true and one is false.

(*b*) if x = 3 and y = 3

```
(x > 0) && (y == 4)
  T     |       |
        |       F
        F
```

Evaluate this expression first.
Evaluate this expression second.
Finally follow the rules for AND, false if one side is true and one is false.

EXAMPLE 4.6 What is the value of each of these expressions given the values of x and y?

(*a*) if x = −3 and y = 4

```
(x > 0) || (y == 4)
  F     |       |
        |       T
        T
```

Evaluate this expression first.
Evaluate this expression second.
Finally follow the rules for OR, true if one side is true and one is false.

(*b*) if x = 3 and y = 3

```
(x > 0) || (y == 4)
  T     |       |
        |       F
        T
```

Evaluate this expression first.
Evaluate this expression second.
Finally follow the rules for OR, true if one side is true and one is false.

EXAMPLE 4.7 What is the value of each of these expressions given the values of x and y?

(*a*) if x = −3 and y = 4

```
(x < 0) || (x == 3) && (y > 0)
  T     |       |       |       |
        |       F       |       |
        |               |       T
        |               F
        T
```

Evaluate this expression first.
Evaluate this expression second.
Evaluate this expression third.
Evaluate AND before OR, false if either is false.
Finally follow the rules for OR, true if one side is true and one is false.

(*b*) if x = 3 and y = 3

```
(x > 0) || (x == 3) && (y > 0)
  T     |       |       |       |
        |       T       |       |
        |               |       T
        |               T
        T
```

Evaluate this expression first.
Evaluate this expression second.
Evaluate this expression third.
Evaluate AND before OR, true if both are true.
Finally follow the rules for OR, true if one side is true and one is false.

EXAMPLE 4.8 What is the value of each of these expressions given the values of x and y?

(*a*) `if x = 3`
`(x > 0) && (x < 100)`

```
  T     |       |
        |       T
        T
```

Evaluate this expression first.
Evaluate this expression second.
Finally follow the rules for AND, true if both are true.

Note: The algebraic expression (0 < x < 100) cannot be used. Each Boolean expression for x must be tested separately.

(*b*) `if x = 3`
`!(x > 0) && (x < 100)`

```
  T     |       |
F       |       |
        |       T
        F
```

Evaluate this expression first.
Evaluate this expression second.
Evaluate this expression third.
Finally follow the rules for AND, true ONLY if both sides are true.

(*c*) `if x = 3`
`!((x > 0) && (x < 100))`

```
| T     |       |
|       |       T
|       T
F
```

Evaluate this expression first.
Evaluate this expression second.
Third, follow the rules for AND, true if both are true.
Finally, because of the parentheses, the NOT is evaluated last, and changes the final value.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 4.1

4.1 CÁC BIỂU THỨC BOOLE

George Boole nhà toán học sống vào thế kỷ thứ 19, đã nghiên cứu về mối quan hệ logic giữa con người và toán học. Ông ta đã thành lập ngành đại số Boole, được tạo mô hình theo các mẫu suy tư của con người. Ngành đại số này được dùng rộng rãi ngày nay để thiết kế các mạch máy tính nơi mà tất cả các biến chỉ có các giá trị zero hoặc một. Các biểu thức Boole bản thân nó có thể là một hằng hoặc nó có thể là các hằng hoặc các biến được nối kết bởi cac toán tử quan hệ hoặc bằng. Các biểu thức Boole riêng lẻ có thể được kết hợp thành câu lệnh phức tạp hơn bằng cách nối kết chúng với toán tử logic NOT, AND vá OR.

4.1.1 Các biến Boole và các hằng

Một biểu thức Boole có thể là một biến Boole hoặc một hằng. Các biến như thế thường được gọi là các cờ bởi vì chúng có thể liên quan đến tình trạng của các điều kiện trong chương trình. Điều quan trọng là

ta phải cung cấp cho chương trình này các tên mang tính mô tả bởi vì các tên mang tính mô tả giúp cho chúng ta hiểu và gỡ rối các chương trình. Các tên biến mang tính mô tả có thể có chứa dataOK workDone hoặc itemFound. Một vài ngôn ngữ cũng có kiểu dữ liệu Boole được tạo sẵn. Tuy nhiên thông thường các biến nguyên được dùng để trình bày các điều kiện Boole. Ví dụ trong một vài ngôn ngữ, zero (0) ám chỉ sai và các giá trị khác zero ám chỉ đúng. Trong C, C++ và Java bất cứ giá trị nào khác zero đều ám chỉ đúng, nhưng thường giá trị một (1) được dùng. Trong Visual Basic , trừ 1 (-1) ám chỉ đúng.

***VÍ DỤ 4.1** Minh họa các biến Boole sau đây theo C, C++ và Java.*

```
const TRUE=1;
const FALSE=0;
int dataOK, workDone, itemFound;
dataOK=FALSE; // set to false, data is NOT OK
workDone=TRUE;// set to true, the work is completed
itemFound=FALSE;// set to false, the item was not found
```

4.1.2 Các biểu thức quan hệ

Các biểu thức Boole có thể được tạo bằng cách nối kết các hằng hoặc các biến bởi một toán tử quan hệ. Các toán tử quan hệ và các toán tử bằng như được biểu thị trong bảng 4.1 Tương đương tiếng Anh của chúng được dùng dưới dạng cách đọc thông thường và dễ hiểu.

***Bảng 4.1** Toán tử quan hệ*

Các phép tính số học luôn luôn được thực hiện trước khi các toán tử

Relational Operators in C, C++ and JAVA	Relational Operators in Visual Basic	English Equivalent
<	<	less than
<=	<=	less than or equal to
>	>	greater than
>=	>=	greater than or equal to
Equality Operators	Equality Operators	
==	=	equal to
!=	<>	not equal to

quan hệ và toán tử bằng được lượng giá. Ví dụ trong trong biểu thức

$$4 + 5 < 20$$

4 và 5 sẽ được cộng cho nhau trước khi kiểm tra xem thử kết quả có

nhỏ hơn 20 hay không. Tất cả các phép tính và phép so sánh được thực hiện từ trái sang phải ngoại trừ được chỉ định theo một cách khác thông qua việc sử dụng các dấu móc đơn. Các biến và/hoặc các hằng được nối kết với nhau với nhau bởi các toán tử quan hệ và toán tử bằng phải luôn luôn có cùng kiểu. Ví dụ khảo sát các biểu thức quan hệ sau đây trong bảng 4.12. Trong cột thứ hai cả x và y đều được khai báo theo kiểu nguyên. Trong cột thứ ba cả hai đều được khai báo theo kiểu ký tự.

***Bảng 4.2** Các ví dụ về biểu thức Boole*

Boolean Expression	Value if x = 3 and y = 4	Value if x = 'b' and y = 'b'	Explanation
x > y	false	false	If x and y are the same, x is not greater than y
x == y	false	true	They are precisely equal to each other
x != y	true	false	They are precisely not equal to each other
x < y	true	false	If x and y are the same, x is not less than y
x <= y	true	true	If they are the same, then x is either less than or equal to y

***VÍ DỤ 4.2** Tìm giá trị của mỗi một biểu thức trong số các biểu thức được cho sau đây, biết rằng giá trị x và y đã cho?*

Boolean Expression	Value if x = 3 and y = 4	Explanation
x - 4 < y	true	(-1) is less than 4
x + 1 == y	true	4 is precisely equal to 4
x != y - 1	false	It is false that 3 is precisely not equal to 3
x + 1 >= y	true	3 is greater than or equal to 3

***VÍ DỤ 4.3** Các biến chuỗi cũng có thể được so sánh. Trong trường hợp này máy tính sẽ xem xét các giá trị từ trái sang phải, theo lần lượt từng ý tự và so sáng ASCII của mỗi ký tự Hãy khảo sát các biểu thức Boole trong bảng 4.3.*

Bảng 4.3

Boolean Expression	Value	Explanation
"Ann" < "Annette"	true	The 4th character is considered to be null, which in ASCII is zero, and less than the ASCII value of any other character
"Sam" == "Sam"	true	Each character is precisely the same
"Johnson" <= "Johnsen"	false	The 6th character 'e' is less than 'o', which makes the entire string "Johnsen" less than the entire string "Johnson"

4.1.3 Các biểu thức logic

Các biến Boole riêng biệt hoặc các biểu thức có thể được kết hợp thành các mệnh đề phức tạp hơn bằng cách nối kết chúng với các toán tử logic. Các toán tử logic dùng cho nhiều ngôn ngữ được minh họa theo thứ tự ưu tiên ở bảng 4.4 với phần giải thích tiếng Anh. Quy trình căn bản để lượng giá các biểu thức Boole phức là:

(1) Lượng giá tất cả các biểu thức quan hệ riêng biệt từ trái sang phải.

(2) Lượng giá các biểu thức logic trong các dấu ngoặc đơn.

(3) Lượng giá các biểu thức logic theo thứ tự ưu tiên.

***Bảng 4.4** Toán tử logic*

Logical Operators in C, C++, and Java	English and Visual Basic	Explanation	Associativity
!	NOT	has single operand, and returns its negation (opposite)	Left to right
&&	AND	returns true ONLY if BOTH operands are true, false if EITHER or BOTH operands are false	Left to right
||	OR	returns true if EITHER or BOTH operands are true, false ONLY if BOTH operands are false	Left to right

Bảng 4.5 minh họa thứ tự ưu tiên trong tất cả các toán tử, bao gồm cả toán tử số học đã được giải thích trong chương 3. Mỗi phần của biểu thức Boole được lượng giá theo thứ tự được chỉ định.

***Bảng 4.5** Thứ tự ưu tiên*

Operators	Operation	Order of Precedence	Associativity
( )	Parentheses	1	
^ (VB only)	Exponentiation	2	
! or NOT	Negation	3	Left to right
* /	Multiplication and division	4	Left to right
\(VB only)	Integer division	5	Left to right
% or MOD	Modulus	6	Left to right
+ −	Addition and subtraction	7	Left to right
< <= > >=	Relational operators	8	Left to right
== != or <>	Equality operators	9	Left to right
&& or AND	Logical AND	10	Left to right
|| or OR	Logical OR	11	Left to right

VÍ DỤ 4.4 *Tìm giá trị của mỗi một biểu thức sau đây ứng với x và y đã cho? Lưu ý, mỗi một biểu thức Boole được lượng giá trước tiên phụ thuộc vào dấu ngoặc đơn. Sau đó, các tổ hợp Boole kết quả được lượng giá kế tiếp theo trình tự biểu đồ ưu tiên. Cuối cùng là biểu thức tổng thể được lượng giá.*

(*a*) **if x = −3 and y = 4**

(x == −3) && (y != 3)

T	| |	Lượng giá biểu thức này trước tiên x là – 3
	| T	Lượng giá biểu thức này kế tiếp y không bằng 3
	T	Sau cùng lượng giá các quy tắc dành cho AND, đúng nếu cả hai đều đúng.

(*b*) **if x = 3 and y = 3**

(x == −3) && (y != 3)

F	| |	Lượng giá biểu thức này trước tiên
	| F	Lượng giá biểu thức này thứ hai
	F	Cuối cùng tuân theo các quy tắc dùng cho AND, sai nếu cả hai cùng sai.

VÍ DỤ 4.5 *Tìm giá trị của mỗi một biểu thức ứng với những x và y đã cho? Lưu ý mỗi một biểu thức Boole được lượng giá trước tiên, phụ thuộc vào các dấu ngoặc đơn. Sau đó là các tổ hợp Boole kế tiếp theo biểu đồ trình tự ưu tiên và cuối cùng là biểu thức tổng thể được lượng giá.*

(*a*) **if x = −3 and y = 4**

(x > 0) && (y == 4)

F	| |	Lượng giá biểu thức này trước tiên.
	| T	Lượng giá biểu thức này kế tiếp.
	F	Cuối cùng tuân theo các quy tắc dành cho AND, sai nếu một vế là đúng, một vế là sai.

(*b*) **if x = 3 and y = 3**

(x > 0) && (y == 4)

T	| |	Lượng giá biểu thức này trước tiên.
	| F	Lượng giá biểu thức này kế tiếp.
	F	Cuối cùng tuân theo các quy tắc dành cho AND, sai nếu một vế là đúng, một vế là sai.

VÍ DỤ 4.6 *Tìm giá trị của mỗi biểu thức ứng với giá trị của x và y đã cho*

(*a*) **if x = −3 and y = 4**

(x > 0) || (y == 4)

F	| |	Lượng giá biểu thức này trước tiên
	| T	Lượng giá biểu thức này thứ hai
	T	Cuối cùng tuân theo quy tắc OR, đúng nếu mỗi vế là đúng và vế kia là sai.

(b) **if x = 3 and y = 3**

(x > 0) || (y == 4)

T	|	Lượng giá biểu thức này trước tiên.
|	F	Lượng giá biểu thức này thứ hai.
T		Cuối cùng tuân theo các quy tắc dành cho OR, đúng nếu một vế là đúng, và một vế là sai.

VÍ DỤ 4.7 *Tìm giá trị của mỗi biểu thức trong các biểu thức sau đây ứng với giá trị x và y?*

(a) **if x = −3 and y = 4**

(x < 0) || (x == 3) && (y > 0)

T	|	|	|	Lượng giá biểu thức này đầu tiên.
|	F	|	|	Lượng giá biểu thức này thứ hai.
|	|	|	T	Lượng giá biểu thức này thứ ba.
|	|	F		Lượng giá AND trước OR, sai nếu mỗi vế là sai.
T				Cuối cùng tuân theo các quy tắc OR, đúng nếu mỗi vế là đúng và vế kia là sai.

(b) **if x = 3 and y = 3**

(x > 0) || (x == 3) && (y > 0)

T	|	|	|	Lượng giá biểu thức này trước tiên.
|	T	|	|	Lượng giá biểu thức này thứ hai.
|	|	|	T	Lượng giá biểu thức này thứ ba.
|	|	T		Lượng giá AND trước OR, đúng nếu cả hai đều đúng.
T				Sau cùng quy tắc này dành cho OR, đúng nếu một vế là đúng và vế kia là sai.

VÍ DỤ 4.8 *Tìm giá trị của mỗi trong số các biểu thức sau đây với các giá trị x và y đã cho?*

(a) **if x = 3**

(x > 0) && (x < 100)

T	|	Lượng giá biểu thức này trước tiên.
|	T	Lượng giá biểu thức này thứ hai.
T		Sau cùng tuân theo các quy tắc dành cho AND, đúng nếu cả hai cùng đúng.

Lưu ý: Biểu thức đại số (0<x<100) không thể được dùng. Mỗi biểu thức Boole ứng với x phải được thử nghiệm riêng biệt.

(*b*) `if x = 3`
`!(x > 0) && (x < 100)`

```
  T     |       |
F       |       |
        |       T
        F
```

Lượng giá biểu thức này trước tiên.

Lượng giá biểu thức này thứ hai.

Lượng giá biểu thức này thứ ba.

Cuối cùng tuân theo các quy tắc dành cho AND, chỉ đúng nếu cả hai vế cùng đúng.

(*c*) `if x = 3`
`!((x > 0) && (x < 100))`

```
|  T    |       |
|       |       T
|       T
F
```

Lượng giá biểu thức này đầu tiên.

Lượng giá biểu thức này thứ hai.

Thứ ba là tuân theo các quy tắc dành cho AND đúng nếu cả hai cùng đúng.

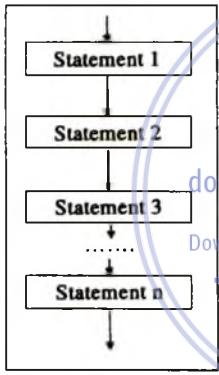
Sau cùng do bởi các dấu ngoặc đơn NOT được lượng giá sau cùng, và thay đổi giá trị cuối cùng.

CHỦ ĐIỂM 4.2

CONTROL STRUCTURES - DEFINITIONS
Các cấu trúc điều khiển - các định nghĩa

Computer programs only do what the programmer has told them to do. They execute the instruction statements in three ways: sequence, selection, and repetition. Most of the time, instructions are executed in sequence, one after the other, in the order in which they are written. Figure 4-1 shows the flowchart for sequence.

EXAMPLE 4.9 What will be the value *of* the variable *c* after this code section?




```
Code                Pseudocode
a=4;                //set a to 4
b=6                 //set b to 6
a=10;               //set a to 10 instead of 4
c=a+b;              //add the two variables together
```

Fig. 4.1 Sequence *flowchart*.

Each statement is executed in order, so the final value *of* the variable c would equal 16, because the variable a was changed before the addition.

Sometimes, the programmer has two or more possible instructions for the computer to execute, with the decision to be made at runtime from the value of a variable or a Boolean expression. In this case, the programmer uses **selection.** The computer selects only ONE instruction (or set of instructions) to execute from two possible choices depending upon whether the Boolean expression is true or false, or from several possible choices depending upon the value of the specified variable.

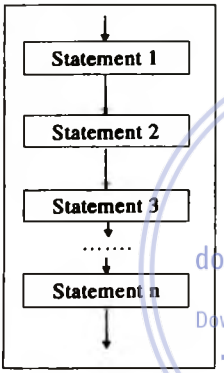
Often, the programmer wants to execute one or more instructions over and over, or perform **repetition.** Once again, a Boolean expression helps to determine how many times the instruction (or set of instructions) will execute.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 4.2

4.2 CÁC CẤU TRÚC ĐIỀU KHIỂN – ĐỊNH NGHĨA

Các chương trình máy tính chỉ thực hiện những gì mà người lập trình báo chúng phải làm. Chúng thực thi các câu lệnh chỉ dẫn theo ba cách: tuần tự, chọn lựa và lặp lại. Hầu hết mọi lúc, các lệnh được thực thi theo một tuần tự lệnh này sang lệnh khác, thứ tự mà qua đó chúng được viết. Hình 4.1 biểu thị lưu đồ dành cho chuỗi thứ tự này.

VÍ DỤ 4.9 *Giá trị của biến c sau phần mã này sẽ là bao nhiêu?*



Mã	*Mã giả*
`a=4;`	*//set a to 4*
`b=6`	*//set b to 6*
`a=10;`	*//set a to 10 instead of 4*
`c=a+b;`	*//add the two variables together*

Hình 4.1 *Lưu đồ tuần tự*

Mỗi câu lệnh được thực thi theo thứ tự, vì thế giá trị của biến c sẽ bằng 16 bởi vì biến a bị thay đổi trước khi cộng.

Đôi khi người lập trình lại có hai hoặc nhiều lệnh để máy tính phải thực thi, với quyết định phải được thực hiện tại thời gian chạy tại giá trị của một biến hoặc một biểu thức Boole. Trong trường hợp này người lập trình phải sử dụng chọn lựa. Máy tính chỉ chọn lựa một lệnh (hoặc một tập hợp các lệnh) để thực thi. Từ hai chọn lựa có thể có phụ thuộc vào trường hợp biểu thức Boole là đúng hay sai hoặc phụ thuộc vào trường hợp các chọn lựa có thể có chỉ phụ thuộc vào giá trị của biến đã được chỉ định.

Thông thường người lập trình muốn thực thi một hoặc nhiều lệnh hoặc thực thi phép lặp lại. Một lần nữa, công thức Boole giúp để xác định lệnh (hoặc tập hợp các lệnh) sẽ thực thi bao nhiêu lần.

CHỦ ĐIỂM 4.3

SELECTION
Sự lựa chọn

Selection allows the computer to choose the instructions to execute. Other instructions may be ignored. A choice may be made between two alternatives, or from many possible alternatives. For readability, the statements within each alternative are usually indented.

4.3.1 Choosing Between Two Alternadves

The computer can choose between two possible alternatives, although the second alternative may just be to ignore the first statement. For example, you might say, "If it is raining, I'll take my umbrella." Implied is the option to ignore the umbrella if it is not raining. Or you might say, "If it is cold outside, I'll drink hot tea. Otherwise, I'll have ice water." The option to drink something is not ignored; there are two clear options from which to choose. Figure 4-2 illustrates the flowcharts for the two kinds of statements.

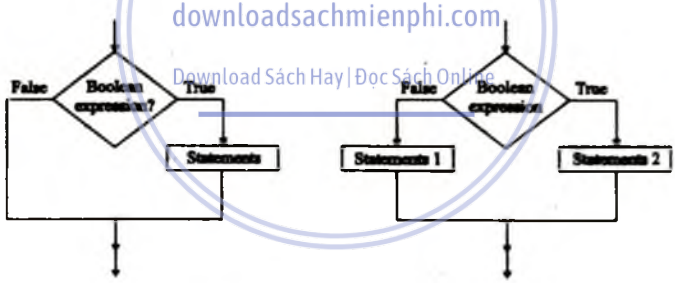


Fig. 4-2 Single "If" and "If...Else" structures.

Most programming languages have single "If" statements, as well as "If ..Else" statements. Specific forms are shown in Table 4-6. In C, C++, and Java, the braces may be omitted if there is only one statement.

If the Boolean expression is true, the first set of statements will be executed. If the Boolean expression is false, in the single "If" structure, nothing happens, and the control passes to the statements following the "If". In the "It -Else" structure, if the Boolean expression is false, the Else statements will execute.

Table 4-6 If and If...Else Statements.

C, C++, and Java	Visual Basic

`if (Boolean expression)`
`{ statement(s)`
`}`
`if (Boolean expression)`
`{ statement(s)`
`}`
`else`
`{ statement(s)`
`}`
	`If Boolean expression Then`
	`statement(s)`
	`End If`
	`If Boolean expression Then`
	`statement(s)`
	`Else`
	`statement(s)`
	`End If`

In Example 4.9 below, notice the difference in the equality testing between C++ and Visual Basic. C, C++, and Java require the double equal sign "==" to test equality. However, if a single equal sign is used no error message is given. The computer assumes it is an assignment statement, puts the value on the right into the variable on the left, and then tests whether the result is zero or non-zero. In debugging a program, one of the first things to look for is the correct use of the double equal sign. In addition, Visual Basic requires that the word "Else" be on a line by itself

EXAMPLE 4.9 Write an "If" statement that will set a character variable called message to "T" if the Boolean variable *raining* is true. Notice the indentation of the statements.

C, C++, and Java:

```
if (raining==true) //'if(raining=true)'wouldassignthevaluetrue,
   message='T'; // and always execute the if statements
```

Visual Basis:

```
If (raining=true) Then
  message="T"
End If
```

Each language has rules about placement of the control statements. In Visual Basic, as stated previously, the "Else" should always be on its own separate line with the statements to be executed below. In C, C++, and Java, if there is only one statement to execute for each condition, they may be placed on the same line as the "if" and "else." If more than one statement is required, brackets must be used. In that case, the statements within each alternative are indented.

EXAMPLE 4.10 Write an "If..Else" statement that will set a character variable called *drink* to "H" for (Hot Tea) if the Boolean variable cold is true, or "W" for (Ice Water) otherwise. Show how the code would be different if the running total of each kind of drink was to be tabulated.

One statement for each alternative:

C, C++, and Java:
```
if (cold==true) drink='H';
 else drink='W';
```

Visual Basic:
```
If (cold=true) Then
    drink='H'
Else
    drink='W'
End If
```

More than one statement for each alternative:

C, C++, and Java:
```
if (cold==true)
{
    drink='H';
    hot++;
}
else
{
    drink='W';
    warm++;
}
```


Visual Basic:
```
If (cold=true) Then
    drink='H'
    hot=hot+1
Else
    drink='W'
    warm=warm+1
End If
```

43.2 Choosing One From Several Alternatives

Sometimes there are more than two alternatives from which to choose. For example, if it is Monday or Friday, you have class. On Tuesday and Thursday you have lab. Wednesday you work, and Saturday you sleep late. Your activity depends upon which day it is. Most modern languages have two ways to indicate such a choice: a series of "If' statements, or a more efficient case structure. The flowcharts for both structures are illustrated in Fig. 4-3.

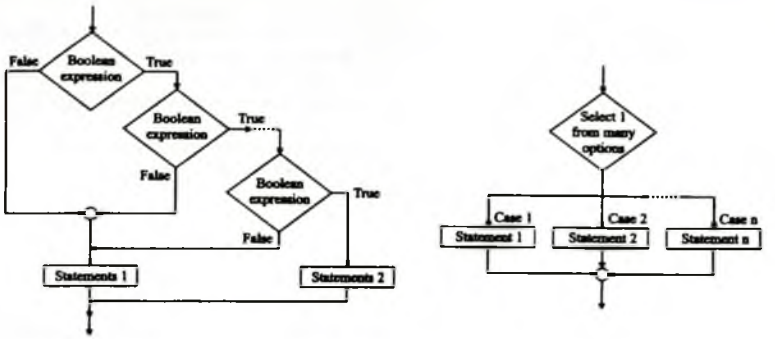


Fig. 4.3 Nested "If" and Case structures.

The case structure examines a variable or expression, and allows the choices to match its possible values. Usually, a default is specified in case none of the possible values is present. The alternate structures for all three languages are shown in Table 4-7.

Table 4-7 Nested Ifs and Cue Structures.

C, C++, and Java	Visual Basic

`if (Boolean Expression)`
`...{ statements }`
`else {`
`  if (Boolean Expression)`
`...  { statements }`
`  else  {`
`  if (Boolean Expression)`
`  ...{ statements }`
`  else`
`... { statements }`
`  }`
`}`
	`If Boolean Expression Then`
	`  Statement(s)`
	`Else`
	`  If Boolean Expression Then`
	`  Statement(s)`
	`  Else`
	`  If Boolean Expression Then`
	`  Statement(s)`
	`... Else`
	`  Statement(s)`
	`  End If`
	`  End If`
	`End If`

C, C++, and Java	Visual Basic

`if (Boolean Expression)`
`{`
`  statement(s)`
`}`
`else if`
`{`
`  statement(s)`
`}...`
`else`
`{`
`  statement(s)`
`}`
	`If Boolean Expression Then`
	`  Statement(s)`
	`ElseIf Boolean Expression Then`
	`  Statement(s)`
	`ElseIf Boolean Expression Then`
	`  Statement(s)`
	`...`
	`End If`

`switch (variable or expression)`
`{`
`  case (1st value): statement(s);`
`  break;`
`  case (2nd value): statement(s);`
`  break;`
`...`
`  default : statement(s);`
`}`
	`Select Case variable or expression`
	`  Case (1st value)`
	`  statement(s)`
	`  Case (2nd value)`
	`  statement(s)`
	`...`
	`  Case Else`
	`  statement(s)`
	`End Select`

It is possible to convert a complex "If ..Else" statement to a case structure. Often the logic is more evident using the switch or select case statement, especially if there is a compound condition. Alternate versions are shown in Examples 4.1,1 and*12. Notice the awkward nature of nested!ffs-in the first version. These three versions demonstrate how much easier the camstivcture is to write and to understand. The Visual Basic equivalents are shown in Solved Problem 4.4 at the end of the chapter.

EXAMPLE 4.11 Write a section of code that will take in a number 1 through 7, and print out the day of the week.

"If...Else" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1)
      cout<<'Sunday'<<endl;
else {
      if (dayNum==2)
            cout<<'Monday'<<endl;
      else {
            if (dayNum==3)
                  cout<<'Tuesday'<<endl;
            else {
                  if (dayNum==4)
                        cout<<'Wednesday'<<endl;
                  else {
```

```
                    if (dayNum==5)
                          cout<<'Thursday'<<endl;
                    else {
                          if (dayNum==6)
                                cout<<'Friday'<<endl;
                          else {
                                if (dayNum==7)
                                      cout<<'Saturday'<<endl;
                                else(
                                      cout<<'Only 1 through 7'<<endl;
                                )
                          }
                    }
              )
        )
  )
```

"If... Elseif" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1)
      cout<<'Sunday'<<endl;
else if (dayNum==2)
      cout<<'Monday'<<endl;
else if (dayNum==3)
      cout<<'Tuesday'<<endl;
else if (dayNum==4)
      cout<<'Wednesday'<<endl;
else if (dayNum==5)
      cout<<'Thursday'<<endl;
else if (dayNum==6)
      cout<<'Friday'<<endl;
else if (dayNum==7)
      cout<<'Saturday'<<endl;
else
      cout<<'Only 1 through 7'<<endl;
```


Case Structwe version: The variable dayNum is the variable tested for each case.

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
switch (dayNum)
{
      case 1: cout<<'Sunday'<<endl; break;
      case 2: cout<<'Monday'<<endl; break;
      case 3: cout<<'Tuesday'<<endl; break;
      case 4: cout<<'Wednesday'<<endl; break;
      case 5: cout<<'Thursday'<<endl; break;
      case 6: cout<<'Friday'<<endl; break;
      case 7: cout<<'Saturday'<<endl; break;
      default: cout<<'Only 1 through 7'<<endl;
```

EXAMPLE 4.12 Write a section of code that will determine whether a day of the week is a working day or a weekend.

"If..Else" version

```
cout<<endl<<"Enter a day of the week - 1 for Sun, etc."; cin>>dayNum;
if (dayNum==1 || dayNum==7)
      cout<<"Weekend - have fun"<<endl;
else
      if (dayNum>=2 && dayNum<=6)
            cout<<"Weekday - go to work"<<endl;
      else
            cout<<"Only 1 through 7"<<endl;
```

" If ... Elseif" version

```
cout<<endl<<"Enter a day of the week - 1 for Sun, etc."; cin>>dayNum;
if (dayNum==1 || dayNum==7)
      cout<<"Weekend - have fun"<<endl;
else if (dayNum>=&& dayNum<=6)
      cout<<"Weekday - go to work"<<endl;
else
      cout<<"Only 1 through 7"<<endl;
```


Case Structure version: Notice the cases can be combined. Each line will be executed sequentially until it reaches the *break* command.

```
cout<<endl<<"Enter a day of the week - 1 for Sun, etc."; cin>>dayNum;
switch (dayNum)
{
      case 1:
      case 7: cout<<"Weekend - have fun"<<endl; break;
      case 2:
      case 3:
      case 4:
      case 5:
      case 6: cout<<"Weekday - go to work"<<endl; break;
      default: cout<<"Only 1 through 7"<<endl;
}
```

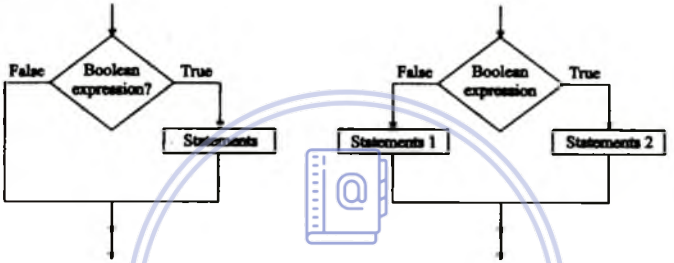
HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 4.3

4.3 CHỌN LỰA

Phương pháp thực thi theo kiểu chọn lựa cho phép máy tính chọn các lệnh để thực thi. Các lệnh khác có thể bị bỏ qua . Một phần chọn lựa có thể được tạo ra giữa hai biến tuần tự hoặc nhiều biến tuần tự. Để dễ đọc, các câu lệnh trong mỗi biến tuần tự thường được thụt vào.

4.3.1 Chọn giữa hai tình huống có thể có

Máy tính có thể chọn giữa hai tình huống có thể có, mặc dù tình huống thứ hai đôi khi bỏ qua câu lệnh đầu tiên. Ví dụ bạn có thể nói. "Nếu trời mưa thì tôi sẽ mang theo dù". Câu này ngụ ý rằng bạn sẽ bỏ qua việc mang dù nếu trời không mưa hoặc có thể nói rằng "Nếu trời lạnh, tôi sẽ uống một tách trà Ngược lại tôi sẽ uống một ly nước mát ". Tùy chọn muốn điều gì đó không bị bỏ qua; có hai tùy chọn rõ ràng mà từ đó bạn chọn. Hình 4.2 minh họa lưu đồ của hai kiểu câu lệnh này.



***Hnh 4.2** Các cấu trúc "If" và "If .. Else"*

Hầu hết các ngôn ngữ lập trình đều có các câu lệnh "If" , cũng như "If ... Else". Các dạng đặc biệt được minh họa trong bảng 4.6. Trong C, C++ và Java , các dấu móc có thể được bỏ qua nếu chúng chỉ có một câu lệnh.

***Bảng 4.6** Các câu lệnh If và If .. Else*

C, C++, and Java	Visual Basic

`if (Boolean expression)`
`{ statement(s)`
`}`
`if (Boolean expression)`
`{ statement(s)`
`}`
`else`
`{ statement(s)`
`}`
	`If Boolean expression Then`
	`statement(s)`
	`End If`
	`If Boolean expression Then`
	`statement(s)`
	`Else`
	`statement(s)`
	`End If`

Nếu câu lệnh Boole là đúng thì tập hợp các câu lệnh đầu tiên sẽ được thực thi. Nếu biểu thức Boole sai, trong cấu trúc "If", không có điều gì xảy ra , mục điều khiển sẽ chuyển sang câu lệnh theo sau cấu trúc "If".

Trong câu lệnh "If ... Else", nếu biểu thức Boole là sai thì các câu lệnh Else sẽ thực thi.

Trong ví dụ 4.9 dưới đây, lưu ý sự khác biệt giữa các thử nghiệm máy tính bằng nhau giữa C++ và Visual Basic. C, C++ , và Java thì yêu cầu phải có một dấu bằng đôi "==" để thử nghiệm tính bằng nhau. Tuy nhiên, nếu có một dấu bằng được dùng thì không có phản biệt lỗi nào xảy ra . Máy tính giả sử đây là một câu lệnh gán, nó sẽ đặt giá trị nằm phía bên phải vào biến phía bên trái rồi thử nghiệm xem thử kết quả bằng 0 hay khác 0. Trong việc gỡ rối một chương trình, một trong những điều cần làm đó là phải xem xét cách sử dụng đúng dấu bằng kép. Ngoài ra, Visual Basic yêu cầu rằng từ "Else" phải nằm trên chính hàng của nó.

***VÍ DỤ 4.9** Viết một câu lệnh "If" mà nó sẽ xác lập một biến ký tự có tên là message sang "T" nếu biến Bool raining là đúng. Lưu ý sự thụt dòng của các câu lệnh.*

C, C++, and Java:

```
if (raining==true) // 'if(raining=true)' would assign the value true,
   message='T'; // and always execute the if statements
```

Visual Basis:

```
If (raining=true) Then
  message="T"
End If
```


Mỗi ngôn ngữ đều có quy tắc về vị trí của các câu lệnh điều khiển. Trong Visual Basic như phát biểu trước đây, "Else" sẽ luôn luôn nằm trên dòng riêng biệt của chính nó với các câu lệnh được thực thi bên dưới. Trong C, C++ và Java, nếu chỉ có một câu lệnh để thực thi cho mỗi điều kiện, thì chúng cũng phải được đặt lên cùng một dòng giống như "If" và "Else" nếu có nhiều câu lệnh cần thiết, thì người ta sẽ sử dụng các dấu móc. Trong trường hợp đó các câu lệnh nằm bên trong mỗi một biến thể đều phải được thụt vào.

***VÍ DỤ 4.10** Hãy viết một câu lệnh "If... Else" để xác lập một biến ký tự có tên là drink sang "H" dành cho (Hot Tea) nếu biến Boolean cold là đúng, hoặc "W" dành cho chữ (Ice Water) nếu rơi vào trường hợp khác. Hãy trình bày mã khác nhau như thế nào nếu tổng chạy cho mỗi kiểu drink được lập thành bảng.*

Một câu lệnh ứng với mỗi biến thể:

C, C++, và Java:
```
if (cold==true) drink='H';
   else drink='W';
```

Visual Basic:
```
If (cold=true) Then
   drink="H"
Else
   drink="W"
End If
```

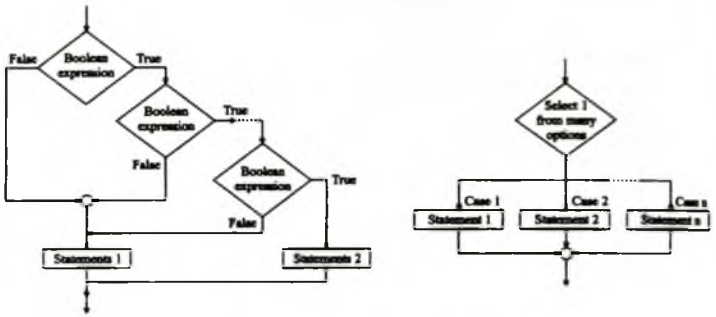
Nhiều câu lệnh dành cho mỗi biến thể:

C, C++, and Java:
```
if (cold==true)
   {
      drink='H';
      hot++;
   }
   else
   {
      drink='W';
      warm++;
   }
```

Visual Basic:
```
If (cold=true) Then
   drink="H"
   hot=hot+1
Else
   drink="W"
   warm=warm+1
End If
```


4.3.2 Chọn một từ nhiều biến thể

Đôi khi có hơn hai biến thể để bạn tự chọn. Ví dụ, nếu nhằm vào ngày thứ Hai hoặc thứ Sáu thì bạn phảo có mặt ở lớp. Còn nếu nhằm ngày thứ Ba và thứ Năm thì bạn phải vào phòng thí nghiệm. Ngày thứ tư bạn làm việc, ngày thứ Bảy bạn ngủ trễ. Hoạt động của bạn phụ thuộc vào ngày. Hầu hết các người lập trình đều có hai cách để xác định một chọn lựa như thế. Một buổi các câu lệnh "If" hoặc cấu trúc trường hợp hữu dụng nhất. Các lưu đồ dành cho hai cấu trúc được minh họa trong hình 4.3.



Hình 4.3 *Các cấu trúc If và Case được lồng nhóm.*

Cấu trúc case xem xét biến hoặc biểu thức rồi cho phép các chọn lựa phải kết hợp với các giá trị có thể có của nó. Thông thường mặc định được chỉ định trong trường hợp không có giá trị nào có mặt. Các cấu trúc biến thể dành cho tất cả ba ngôn ngữ được minh hoạ trong bảng 4.7.

Bảng 4.7 *Các cấu trúc If và Case lồng nhóm.*

C, C++, and Java	Visual Basic

`if (Boolean Expression)`
`...{ statements }`
`else {`
`  if (Boolean Expression)`
`...  { statements }`
`  else  {`
`  if (Boolean Expression)`
`  ...{ statements }`
`  else`
`... { statements }`
`  }`
`}`
	`If Boolean Expression Then`
	`  Statement(s)`
	`Else`
	`  If Boolean Expression Then`
	`  Statement(s)`
	`  Else`
	`  If Boolean Expression Then`
	`  Statement(s)`
	`... Else`
	`  Statement(s)`
	`  End If`
	`  End If`
	`End If`

C, C++, and Java	Visual Basic

if (*Boolean Expression*)
{
  statement(s)
}
else if
{
  statement(s)
}...
else
{
  statement(s)
}
	If *Boolean Expression* Then
	  Statement(s)
	ElseIf *Boolean Expression* Then
	  Statement(s)
	ElseIf *Boolean Expression* Then
	  Statement(s)
	...
	End If

switch (*variable or expression*)
{
case (1st value): statement(s);
  break;
case (2nd value): statement(s);
  break;
...
default : statement(s);
}
	Select Case *variable or expression*
	  Case (1st *value*)
	  statement(s)
	  Case (2nd *value*)
	  statement(s)
	  ...
	  Case Else
	  statement(s)
	End Select

Bạn có thể đổi một câu lệnh "If ... Else" sang một cấu trúc nhạy kiểu chữ. Thường logic mang tính hiển nhiên hơn khi chúng ta sử dụng switch hoặc chọn một câu lệnh nhạy kiểu chữ đặc biệt nếu có điều kiện khác. Các phiên bản biến thể khác được minh họa trong hình 4.11 và 4.12. Lưu ý rằng ản chất kỳ dị của các câu If được lồng nhóm trong phiên bản đầu tiên. Ba phiên bản này minh họa cách mà cấu trúc nhạy ngữ cảnh được viết và được hiểu dễ dàng như thế nào. Các tương đương Visual Basic được minh họa trong Bài tập có lời giải 1.4 nằm ở cuối chương này.

VÍ Ụ 4.11 *Hãy viết một phần mã vốn sẽ nhận một con số từ 1 đến 7, rồi in ngày trong tuần.*

"If ... Else" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1)
      cout<<'Sunday'<<endl;
else {
      if (dayNum==2)
            cout<<'Monday'<<endl;
      else {
            if (dayNum==3)
                  cout<<'Tuesday'<<endl;
            else {
                  if (dayNum==4)
                        cout<<'Wednesday'<<endl;
                  else {
```

```
                        if (dayNum==5)
                              cout<<'Thursday'<<endl;
                        else {
                              if (dayNum==6)
                                    cout<<'Friday'<<endl;
                              else {
                                    if (dayNum==7)
                                          cout<<'Saturday'<<endl;
                                    else{
                                          cout<<'Only 1 through 7'<<endl;
                                    }
                              }
                        }
                  }
            }
      }
```

"If ... Else if" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1)
      cout<<'Sunday'<<endl;
else if (dayNum==2)
      cout<<'Monday'<<endl;
else if (dayNum==3)
      cout<<'Tuesday'<<endl;
else if (dayNum==4)
      cout<<'Wednesday'<<endl;
else if (dayNum==5)
      cout<<'Thursday'<<endl;
else if (dayNum==6)
      cout<<'Friday'<<endl;
else if (dayNum==7)
      cout<<'Saturday'<<endl;
else
      cout<<'Only 1 through 7'<<endl;
```

Case Structure version: The variable dayNum is the variable tested for each case.

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
switch (dayNum)
{
      case 1: cout<<'Sunday'<<endl; break;
      case 2: cout<<'Monday'<<endl; break;
      case 3: cout<<'Tuesday'<<endl; break;
      case 4: cout<<'Wednesday'<<endl; break;
      case 5: cout<<'Thursday'<<endl; break;
      case 6: cout<<'Friday'<<endl; break;
      case 7: cout<<'Saturday'<<endl; break;
      default: cout<<'Only 1 through 7'<<endl;
}
```


VÍ DỤ 4.12 *Hãy viết một phần mã để xác định cho biết ngày trong tuần có phải là ngày làm việc hay là ngày cuối tuần.*

"If...Else" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1 || dayNum==7)
        cout<<'Weekend - have fun'<<endl;
else
        if (dayNum>=2 && dayNum<=6)
                cout<<'Weekday - go to work'<<endl;
        else
                cout<<'Only 1 through 7'<<endl;
```

"If... Else if" version

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
if (dayNum==1 || dayNum==7)
        cout<<'Weekend - have fun'<<endl;
else if (dayNum>=&& dayNum<=6)
        cout<<'Weekday - go to work'<<endl;
else
        cout<<'Only 1 through 7'<<endl;
```

Case version. Notice the cases can be combined. Each line will be executed sequentially until it reaches the *break* command.

```
cout<<endl<<'Enter a day of the week - 1 for Sun, etc.'; cin>>dayNum;
switch (dayNum)
{
        case 1:
        case 7: cout<<'Weekend - have fun'<<endl; break;
        case 2:
        case 3:
        case 4:
        case 5:
        case 6: cout<<'Weekday - go to work'<<endl; break;
        default: cout<<'Only 1 through 7'<<endl;
}
```

CHỦ ĐIỂM 4.4

REPETITION
Phép lặp

As indicated before, repetition allows the program to execute one or more instructions over and over. Each repetition statement is controlled by a loop control variable (lcv) which has an initial and a final value. The lcv begins with its initial value and must be incremented, decremented, or changed in some way during the loop body. Each time the loop is executed, the lcv is compared to the final value to test whether to execute the loop once again. The initial and final values must be of the same data type as the lcv. It is possible in compound conditional statements for the loop to have more than one lcv. The loop body is always indented for ease in reading.

4.4.1 Fixed Repetition Statements

Fixed **repetition** statements are used when you KNOW in advance how many times the loop should be executed. The flowchart for fixed repetition is shown in Fig. 4-4. The lcv is set to the initial value before the loop is entered. Boolean testing of the lcv against the final value is done BEFORE the loop is entered, so if the initial value already reaches the final value the loop will never be entered. When all the statements in the loop body have been executed, the lcv is automatically incremented by the specified amount.

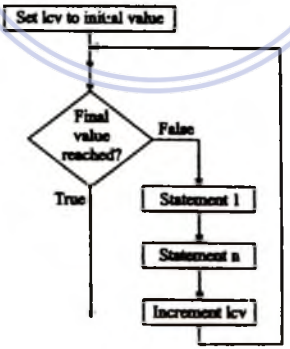


Fig 4.4 Feed repetiition.

Table 4-8 shows the structure of a fixed repetition statement in specific languages. Notice in C, C++, and Java the initial and final value, as well as the amount to increment, are all contained in parentheses separated by

semicolons. In the Visual- Basic version, if the STEP value is omitted the incremental value is set to one by default.

Table 4.8 Fixed Repetition - iv = Initial Value of kv, fv = Final Value of lcv.

C, C++, and Java	Visual Basic

`for (set lcv to iv; test lcv for fv; change lcv by increment value)`
`{`
`  statements`
`}`
	`For lcv=iv TO fv STEP increment value`
	`  Statement(s)`
	`Next lcv`

Here is a sample of a C or'C++ fixed repetition loop that will print the numbers from one to ten. Before the loop begins, the lcv is set to 1, then tested to see if it is less than or equal to 10. After the lcv is printed, lcv is incremented by 1 and then tested again against the value 10. The last time through the loop, the 10 is printed, lcv is incremented to 11, and, since that is greater than 10, the loop stops. Remember, the value of lcv following this loop will be 11.

```
for (lcv=1; lcv <= 10; lcv++)   //BEFORE the loop begins, lcv is set to 1,
                                //then tested to see if it is <= 10
    cout<<lcv<<endl;            //AFTER the lcv is printed, lcv is
                                //incremented by 1 and then tested again
```


The Visual Basic sample of this loop follows the same process. Because the STEP value is left out, lcv is automatically incremented by 1.

```
For lcv=1 To 10       'BEFORE the loop begins, lcv is set to 1, then tested
                      'to see if it is <= 10
  Print lcv
Next lcv              'AFTER the lcv is printed, lcv is incremented by 1
                      'and then tested again
```

The examples below demonstrate the C, C++, and Java versions of fixed repetition loops. Similar examples of Visual Basic loops are shown in exercises at the end of the chapter.

EXAMPLE 4.13 Write a code section to count down from 10 to 0 and then print "Blast off!" In other words, the initial value of the lcv (loop control variable) is 10, the final value that will stop the loop is 0, and the lcv is incremented by (-1) each time through the loop.

```
for (lcv=10; lcv >= 0; lcv--)  //OUTPUT: 10 9 8 7 6 5 4 3 2 1 0 Blast off!
      cout<<lcv<<' ';          //cout the number and then a blank for separation
cout<<"Blast off!"<<endl;
```

EXAMPLE 4.14 Write a section of code to add all the numbers from 0 to 19 and print the sum.

```
sum=0;                                  //initialize the sum to 0
for (num=0; num < 20; num++)            //lcv starts at 0, stops at 20, and increments by 1
      sum+=num;                         //this is the same as sum=sum+num;
cout<<sum<<endl;                        //prints the result, 190
```

NOTE: ++ means add 1 to variable, -- means subtract 1 from variable.

EXAMPLE 4.15 Write a section of code to count by 5's from 0 to 100.

```
for (lcv=0; lcv<101; lcv+=5)    // lcv starts at 0, stops after 100, and increments by 5
      cout<<lcv<<endl;
```

4.4.2 Pretest Repetition Statements

In *pretest loops* the Boolean expression is tested FIRST before the body is executed. Use them when you DON'T KNOW how many repetitions are needed.

- The while loop is executed while the Boolean expression is true; the until loop executes until the Boolean expression becomes true.
- The Boolean expression must be changed in the loop, or it results in an infinite loop.
- The Boolean expression can be an EOF marker, a sentinel value, a Boolean flag, or an arithmetic expression.

The unique characteristics of a pretest loop are:

- The Boolean expression must be given an initial value BEFORE the loop starts.
- If the Boolean expression is initially false, the loop is skipped.

The flowchart for a pretest loop is shown in Fig. 4-5. Notice, like fixed repetition, the testing is done before the loop starts. However, unlike fixed repetition, the lcv must be explicitly changed within the loop. It is not done automatically. Two examples are shown. The first is for the C or C++ while and the Visual Basic Do While and the second demonstrates an alternate version available in Visual Basic, the Do Until.

Notice, the "While" version executes AS LONG AS the Boolean expression is true, and the "Until" version executes UNTIL the Boolean version becomes true. Syntax for these loops is shown in Table 4-9.

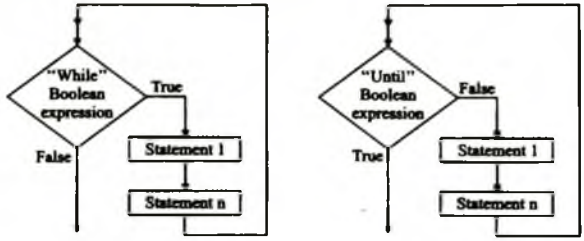


Fig. 4-5 Pretest Loop "While" and "Until".

Table 4-9 Pretest Loops.

C, C++, and Java	Visual Basic	Visual Basic (alternate)

// set lcv to initial value
while (Boolean expression)
{
//statement(s)
//change lcv
}
	'set lcv to initial value
	Do While (Boolean expression)
	'statement(s)
	'change lcv
	Loop
		'set lcv to initial value
		Do Until (Boolean expression)
		'statement(s)
		'change lcv
		Loop

EXAMPLE 4.16 Write a C++ section ofcode to ask the user for a list of numbers and add them until the user wants to stop.

```
sum=0;                                  //initialize sum to 0 to begin
cout<<"Enter an integer, -1 to stop";

cin>>lcv;                               //initial value for lcv
while (lcv !=-1)                        // final value -1 will stop loop
{
   sum+=lcv;
   cout<<"Enter an integer, -1 to stop";

   cin>>lcv;                            //change lcv in loop
}
cout<<sum<<endl;                        //prints the resulting sum
```

Note: if the user enters -1 the first time, the loop will not be entered and the sum will be zero.

EXAMPLE 4.17 Write a C++ section of code to add successive numbers starting with one until the sum is greater than or equal to 1000.

```
num=0;
sum=0;                      //give initial value, in this case sum is the lcv
while (sum<1000)            //sum's final value will be>999
```

```
{
    num++;
    sum+=num;                    //change sum in the loop
}
cout<<sum;                       //sum is 1035 which stops the loop
```

EXAMPLE 4.18 Write two versions of Visual Basic code to implement the same code as Example 4.17.

"While" version:

```
num=0
sum=0
Do While (sum<1000)
     num=num+1
     sum=sum+num
Loop
```

"Until" version

```
num=0
sum=0
Do Until (sum>=1000)
     num=num+1
     sum=sum+num
Loop
```

4.4.3 Posttest Repetition Statements

In **posttest loops** the Boolean expression is tested LAST. Use them when you know you want to execute the loop AT LEAST ONCE (e.g. a menu program). Several characteristics are the same as for a pretest loop.

- The while loop is executed while the Boolean expression is true; the until loop executes until the Boolean expression becomes true.
- The Boolean expression must be changed in the loop, or it results in an infinite loop.
- The Boolean expression can be an EOF marker, a sentinel value, a Boolean flag, or an arithmetic expression.

The unique characteristics of a posttest loop are:

- The Boolean expression may get its initial value within the loop.
- Because it is tested after the loop body has executed, the loop is ALWAYS executed at least once, even if the Boolean expression is false.

The flowchart for a posttest loop is shown in Fig. 4-6. Notice the testing is done after the loop finishes executing. However, unlike fixed repetition, the lcv must be explicitly changed within the loop. It is not done automatically. Two examples are shown. The first is for the GC++ while and the Visual Basic Do While and the second demonstrates an alternate version available in Visual Basic, the Do Until.

Notice, the "While" version executes AS LONG AS the Boolean expression is true, and the "Until" version executes UNTIL the Boolean version becomes true. Syntax for these loops is shown in Table 4-10.

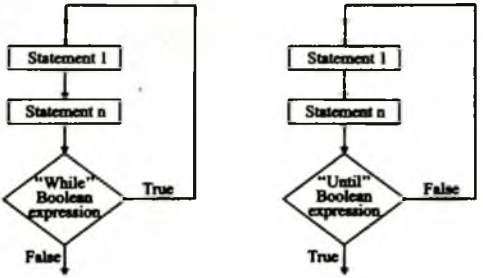


Fig. 4-6 Posttest Loop "**While**" and "**Until**".

Table 4-10 Posttest Loops.

C, C++, and Java	Visual Basic	Visual Basic (alternate)

do {
//statement(s)
//change lcv
} while (Boolean expression);
	Do
	'statement(s)
	'change lcv
	Loop While (Boolean expression)
		Do
		'statement(s)
		'change lcv
		Loop Until (Boolean expression)

EXAMPLE 4.19 Write a section of code to ask the user for a list of numbers and add them until the user wants to stop.

```
sum=0;
lcv=0;                    //give lcv initial value, necessary before add
do                        //loop MUST be entered the first time
{
   sum+=lcv;
   cout<<"Enter an integer, -1 to stop";
   cin>>lcv;              //change lcv in loop
} while (lcv!=-1);        //final value of -1 will stop loop
cout<<sum<<endl;          //prints the result
```

EXAMPLE 4.20 Write a section of code to display a menu of arithmetic choices and have the loop continue until the user chooses to quit.

```
do                                    //loop MUST be entered at least once
{
   cout<<"1. Add"<<endl;
   cout<<"2. Subtract"<<endl;
   cout<<"3. Multiply"<<endl;
   cout<<"4. Divide"<<endl;
   cout<<"5. Quit"<<endl;
   cin>>choice;                       // lcv is allowed to change in loop
   //put in switch statement here to handle calculations
} while (choice!=5);                  //final value of 5 will stop loop
```

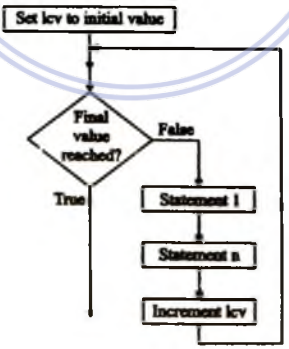
HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 4.4

4.4 PHÉP LẶP

Như đã chỉ định trước đây, phép lặp cho các chương trình thực thi một hoặc nhiều lệnh luôn mãi. Mỗi câu lệnh lặp được điều khiển bởi một biến điều khiển lặp (lcv) nó có một giá trị khởi tạo và một giá trị sau cùng. lcv (b (biến điều khiển lặp) bắt đầu bằng giá trị khởi tạo và phải được tăng hoặc giảm hoặc thay đổi theo một cách thức nào đó trong suốt nội dung lặp. Mỗi khi phép lặp được thực thi, thì lcv được so sánh với giá trị cuối cùng để thử nghiệm xem thử có nên thực thi vòng lặp lại một lần nữa hay không. Các giá trị khởi tạo và sau cùng phải có kiểu dữ liệu giống hệt như lcv. Có thể rằng trong các câu lệnh điều kiện kép vòng lặp có nhiều lcv. Nội dung vòng lặp luôn luôn thụt vào bên trong cho dễ đọc.

4.4.1 Các câu lệnh lặp cố định

Các câu lệnh lặp cố định được dùng lúc bạn biết trước có bao nhiêu lần lặp cần được thực thi. Lưu đồ dành cho phép lặp cố định được minh họa trong hình 4.4. lcv được xác lập sang giá trị khởi tạo trước khi vòng lặp được đưa vào. Phép thử nghiệm Boole của lcv dựa trên giá trị cuối cùng được thực hiện trước khi vòng lặp được đưa vào, vì vậy nếu giá trị khởi tạo đã đạt đến giá trị cuối cùng rồi thì lcv sẽ không bao giờ được nhập vào. Lúc tất cả nội dung các câu lệnh trong vòng lặp được thực thi lcv tự động được gia tăng theo một lượng chỉ định.



***Hình 4.4** Câu lệnh lặp cố định*

Bảng 4.8 trình bày cấu trúc của các câu lệnh lặp cố định theo các ngôn ngữ đặc biệt. Lưu ý trong C, C++ và Java, giá trị khởi tạo và giá trị sau cùng cũng như những gia số luôn luôn được đặt bên trong một

dấu móc đơn và tách rời nhau bằng dấu chấm phẩy. Trong phiên bản Visual Basic, nếu giá trị STEP bị bỏ qua thì giá trị gia số được xác lập sang 1 theo mặc định.

***Bảng 4.8** Phép lặp cố định iv = Giá trị khởi tạo của lcv, fv = giá trị cuối cùng của lcv.*

C, C++, and Java	Visual Basic

for (set lcv to iv; test lcv for fv; change lcv by increment value)
{
statements
}
	For lcv=iv TO fv STEP increment value
	Statement(s)
	Next lcv

Sau đây là mẫu của vòng lặp lặp lại cố định C hoặc C++. Nó sẽ in ra những con số từ 1 cho đến 10. Trước khi vòng lặp bắt đầu, lcv được xác lập sang 1, sau đó được thử nghiệm xem thử nó có nhỏ hơn hay bằng 10. Sau khi lcv được in, lcv gia tăng theo một, sau đó được thử nghiệm lại lần nữa dựa trên giá trị 10. Lần cuối cùng thông qua vòng lặp, 10 được in ra, lev gia tăng sang 11 và bởi vì nó lớn hơn 10 cho nên vòng lặp ngưng. Hãy nhớ rằng giá trị của lcv theo sau vòng lặp này sẽ là 11.

```
for (lcv=1; lcv <= 10; lcv++)   //BEFORE the loop begins, lcv is set to 1,
                                //then tested to see if it is <= 10
   cout<<lcv<<endl;             //AFTER the lcv is printed, lcv is
                                //incremented by 1 and then tested again
```

Mẫu Visual Basic của vòng lặp này tuân theo các quy trình giống nhau. Bởi vì giá trị STEP bị để lại cho nên lcv tự động gia tăng 1.

```
For lcv=1 To 10          'BEFORE the loop begins, lcv is set to 1, then tested
                         'to see if it is <= 10
  Print lcv
Next lcv                 'AFTER the lcv is printed, lcv is incremented by 1
                         'and then tested again
```

Các ví dụ dưới đây minh họa các phiên bàn C, C++ và Java của các vòng lặp cố định. Những ví dụ tươngtự của các vòng lặp Visual Basic cũng được trình bày trong bài tập ở cuối chương.

***VÍ DỤ 4.13** Hãy viết một phần mã để đếm xuống từ 10 cho đến 0 rồi in "Bast off". Nói cách khác giá trị khởi tạo của lcv (biến điều khiển vòng lặp) là 10, giá trị cuối cùng của nó sẽ ngưng vòng lặp là zero và lcv gia tăng một lượng là (- 1) mỗi lần lặp.*

```
for (lcv=10; lcv >= 0; lcv--)  //OUTPUT: 10 9 8 7 6 5 4 3 2 1 0 Blast off!
      cout<<lcv<<' ';          //cout the number and then a blank for separation
cout<<'Blast off!'<<endl;
```

VÍ DỤ 4.14 *Hãy viết một đoạn mã để cộng tất cả các số từ 0 cho đến 19. và in tổng.*

```
sum=0;                          //initialize the sum to 0
for (num=0; num < 20; num++)    //lcv starts at 0, stops at 20, and increments by 1
      sum+=num;                 //this is the same as sum=sum+num;
cout<<sum<<endl;                //prints the result, 190
```

VÍ DỤ 4.15 *Hãy viết một phần mã để đếm theo các bội số của 5 từ 0 cho đến 100.*

```
for (lcv=0; lcv<101; lcv+=5) // lcv starts at 0, stops after 100, and increments by 5
      cout<<lcv<<endl;
```

4.4.2 Các câu lệnh lặp được thử nghiệm trước

Trong phép lặp thử nghiệm trước, biểu thức Boole được thử nghiệm trước tiên trước khi mà nội dung được thựcthi. Hãy sử dụng chúng lúc bạn không biết ấn phải có bao nhiêu vòng lặp.

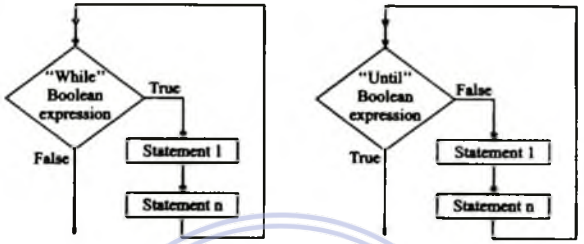
- *Vòng lặp While được thực thi trong khi biểu thức Boole đúng; vòng lặp unti sẽ thực thi cho đến khi biểu thức Boole trở nên đúng.*
- *Biểu thức Boole phải đượcthay đổi trong vòng lặp nếu không thì nó sẽ cho kết quả có một vòng lặp vô hạn.*
- *Biểu thức Bool có thể là một dấu kiểm EOF, một giá trị nhạy ngữ cảnh một cờ Boole hoặc một biểu thức số học.*

Các đặc trưng duy nhất của một vòng lặp thử nghiệm trước là

- *Biểu thức Boole phải đượccho một giá trị khởi tạo trước khi vòng lặp bắt đầu.*
- *Nếu biểu thức Boole khởi đầu bị sai thì vòng lặp được lướt qua.*

Lưu đồ dành cho vòng lặp thử nghiệm trước được minh họa trong hình 4.5. Lưu ý rằng, cũng giống như phép lặp cố định, phép thử nghiệm được thực hiện trước khi vòng lặp bắt đầu. Tuy nhiên, không giống như phép lặp cố định, lcv phải bị thay đổi một cách rõ ràng bên trong vòng lặp. Nó không được thực hiện một cách tự động. Hai ví dụ được minh họa dưới đây. Ví dụ đầu tiên dùng cho C hoặc C++ while và Visual Basic Do While, còn ví dụ thứ hai thì minh họa phiên bản biến đổi có sẵn trong Visual Basic, Do Until.

Lưu ý các phiên bản While thực thi ngay khi biểu thức Boole là đúng, và phiên bản "Until" thực thi cho đến khi phiên bản Boole trở nên đúng. Cú pháp dành cho vòng lặp này được minh họa trong bảng 4.9.



***Hình 4.5** Thử nghiệm vòng lặp "While" và "Until"*

***Bảng 4.9** Các vòng lặp thử nghiệm trước*

C, C++, and Java	Visual Basic	Visual Basic (alternate)

// set lcv to initial value
while (Boolean expression)
{
//statement(s)
//change lcv
}
	'set lcv to initial value
	Do While (Boolean expression)
	'statement(s)
	'change lcv
	Loop
		'set lcv to initial value
		Do Until (Boolean expression)
		'statement(s)
		'change lcv
		Loop

***VÍ DỤ 4.16** Hãy viết một đoạn mã C++ để yêu cầu người dùng về một danh sách những con số rồi cộng chúng cho đến khi người dùng muốn ngưng.*

```
sum=0;                                  //initialize sum to 0 to begin
cout<<"Enter an integer, -1 to stop";

cin>>lcv;                               //initial value for lcv
while (lcv !=-1)                        // final value -1 will stop loop
{
   sum+=lcv;
   cout<<"Enter an integer, -1 to stop";

   cin>>lcv;                            //change lcv in loop
}
cout<<sum<<endl;                        //prints the resulting sum
```

Lưu ý: Nếu người dùng nhập vào số -1 lần đầu tiên, thì vòng lặp sẽ không được đưa vào và tổng sẽ bằng zero.

VÍ DỤ 4.17 *Hãy viết một đoạn mã C++ để cộng các số tuần tự bắt đầu bằng số có thể zero cho đến khi tổng lớn hơn hoặc bằng 1000.*

```
num=0;
sum=0;                    //give initial value, in this case sum is the lcv
while (sum<1000)          //sum's final value will be>999
{
    num++;
    sum+=num;             //change sum in the loop
}
cout<<sum;                //sum is 1035 which stops the loop
```

VÍ DỤ 4.18 *Hãy viết hai phiên bản của mã Visual Basic để thực thi đoạn mã như trong 4.17.*

"While" version:

```
num=0
sum=0
Do While (sum<1000)
      num=num+1
      sum=sum+num
Loop
```

"Until" version

```
num=0
sum=0
Do Until (sum>=1000)
      num=num+1
      sum=sum+num
Loop
```


4.4.3 Các câu lệnh lặp thử nghiệm sau

Trong các vòng lặp thử nghiệm sau, biểu thức Boole được thử nghiệm cuối cùng. Hãy sử dụng chúng lúc bạn biết bạn muốn thực thi vòng lặp ít nhất một lần (ví dụ một chương trình menu). Nhiều đặc trưng giống hệt như là một phép lặp thử trước.

- *Vòng lặp While được thực thi trong khi biểu thức Boole đúng; vòng lặp until thực thi cho đến lúc biểu thức Boole trở thành đúng.*
- *Biểu thức Boole phải được thay đổi trong vòng lặp nếu không thì kết quả của nó nằm trong vòng lặp vô hạn.*
- *Biểu thức Boole có thể là một dấu kiểm EOF, một giá trị nhạy cảm, một cờ Boole hoặc một biểu thức số học.*

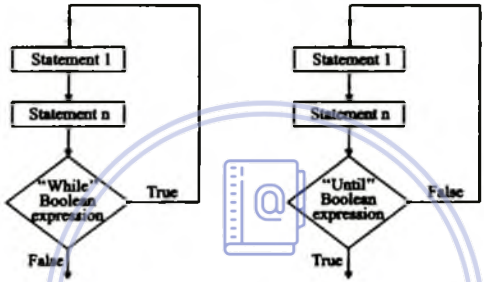
Đặc trưng duy nhất của vòng lặp thử ghiệm sau là:

- *Biểu thức Boole có thể nhận giá trị khởi tạo của nó bến trong vòng lặp.*
- *Bởi vì nó được thử nghiệm sau khi nội dung vòng lặp đã được thực thi cho nên vòng lặp luôn luôn được thực thi ít nhất là một lần thậm chí cả khi biểu thức Boole là sai.*

Lưu đồ dành cho phép lặp thử nghiệm sau được minh họa trong hình 4.6 .Lưu ý phép thử nghiệm được thực hiện sau khi phép lặp hoàn tất

việc thực thi. Tuy nhiên không giống như phép lặp cố định, lcv phải được thay đổi rõ ràng trong vòng lặp. Điều này không được thực hiện một cách tự động. Hai ví dụ được trình bày dưới đây. Ví dụ đầu tiên dành cho C, C++ while và Visual Basic Do While còn ví dụ thứ hai minh họa phiên bản biến thể có sẵn trong Visua Basic, Do Until.

Lưu ý, phiên bản While thực thi ngay khi biểu thức Boole là đúng và phiên bản Until thực thi cho đến khi phiên bản Boole trở nên đúng. Cú pháp dành cho những vòng lặp này được minh họa trong bảng 4.10.



***Hình 4.6** Phép lặp được thử nghiệm sau "While" và "Until"*

***Bảng 4.10** Các phép lặp thử nghiệm sau.*

C, C++, and Java	Visual Basic	Visual Basic (alternate)

do {
//statement(s)
//change lcv
} while (Boolean expression);
	Do
	'statement(s)
	'change lcv
	Loop While (Boolean expression)
		Do
		'statement(s)
		'change lcv
		Loop Until (Boolean expression)

***VÍ DỤ 4.19** Hãy viết một đoạn mã yêu cầu người dùng về một danh sách những con số và cộng chúng cho đến khi người dùng muốn ngưng.*

```
sum=0;
lcv=0;                          //give lcv initial value, necessary before add
do                              //loop MUST be entered the first time
{
    sum+=lcv;
    cout<<'Enter an integer, -1 to stop';
    cin>>lcv;                   //change lcv in loop
} while (lcv!=-1);              //final value of -1 will stop loop
cout<<sum<<endl;                //prints the result
```

VÍ DỤ 4.20 *Hãy viết một đoạn mã hiển thị một menu các chọn lựa số học và có một vòng lặp tiếp tục cho đến khi người dùng muốn thoát.*

```
do                                          //loop MUST be entered at least once
{
    cout<<"1. Add"<<endl;
    cout<<"2. Subtract"<<endl;
    cout<<"3. Multiply"<<endl;
    cout<<"4. Divide"<<endl;
    cout<<"5. Quit"<<endl;
    cin>>choice;                            // lcv is allowed to change in loop
    //put in switch statement here to handle calculations
} while (choice!=5);                        //final value of 5 will stop loop
```

SOLVED PROBLEMS
Bài tập có lời giải

4.1 What is the value of each of these expressions given the values of a, b, and c and of the Boolean variable done? Remember to follow the order of precedence in Table 4-5.

(*a*) if a = −3 and b = 4 and c = 2
(a<c) && (b>c)

```
 T   |     |      Evaluate this expression first.
     |  T  |      Evaluate this expression second.
     T            Finally follow the rules for AND, true if both are true.
```

(*b*) if a = −3 and b = 4 and c = 2
(a<c) || (c>b)

```
 T   |     |      Evaluate this expression first.
     |  F  |      Evaluate this expression second.
     T            Finally follow the rules for OR, true if one or both are true.
```

(*c*) if a = 3 and b = 3 and done = false
(a==−3) || (b!=3) && (!done)

```
 F   |  |  |  |   Evaluate this expression first.
     |  F  |  |   Evaluate this expression second.
     |     |  T   Evaluate this expression third, if done is false, not done is true.
     |     F      AND is evaluated before OR, and follow the rules for AND, true only if both are true.
     F            Finally follow the rules for OR, false if both are false.
```

4.2 What is the value of each of these expressions given the values of a, b, and c and of the Boolean variable done? Remember to follow the order of precedence in Table 4-5.

(*a*) if a = 8 and b = 9 and done is true
(a!=8) && (b!=7) || (!done)

```
 F   |  |  |  |   Evaluate this expression first.
     |  T  |  |   Evaluate this expression second.
     |     |  F   Evaluate this expression third, if done is true, not done is false.
     F            AND is evaluated before OR, and follow the rules for AND, true only if both are true.
     F            Finally follow the rules for OR, false if both sides are false.
```

(*b*) if a = 8 and b = 9 and c = 5
!(a>b) || ((b+c)>=(a−c))

```
 T   |     |      Evaluate this expression first, if a greater than b is false, then the NOT makes it true.
     |  T  |      Evaluate this expression second, 17 is greater than 4.
     T            Finally follow the rules for OR, true if one side is true and one is false, or both sides are true.
```

(c) if a = 8 and b = 9 and c = 5

```
!((a<b) && ((b+c)>=(a-c)))
  |  T    |       |        Evaluate this expression first, a is less than b is true
  |       |       T        Evaluate this expression second, 17 is greater than 4.
  |       T                Follow the rules for AND, true if both sides are true.
  F                        Finally, because of the parentheses, the NOT reverses the value of the entire expression.
```

4.3 Write a relational or logical statement in Visual Basic and GC++ that will express the following:

	Expression
(a)	speed is exactly 65 mph
(b)	name is less than "JONES"
(c)	0<x<100 or, x is between 0 and 100, but not equal to either
(d)	x is NOT between 0 and 100
(e)	exclusive OR - true if x is true and y is false;: or true if x is false and y is true - they are never the same

Solution:

	C/C++	VB
(a)	(speed==65)	(speed = 65)
(b)	(name<"JONES")	(name<"'JONES")
(c)	*(x>o)* && (x<100)	(x>0) AND (x<100)
(d)	*!((x>o) && (x<100))*	*Noar((x>o)* AND (x<100))
(e)	*((x* && !y) | | (!x && y))	((x AND NOT(y)) OR (NOT(x) AND y))

4.4 Write the Visual Basic code for the three kinds of selection structures in Example 4.11.

"If..Else" version

```
'assign dayNum or read in value
If (dayNum=1 OR dayNum=7)
   Print 'Weekend - have fun'
Else
   If (dayNum>=2 AND dayNum<=6)
      Print 'Weekday - go to work'
   Else
      Print 'Only 1 through 7'
   End If
End If
```

"If;-Elseif" version

```
'assign dayNum or read in value
If (dayNum==1 OR dayNum==7)
   Print "Weekend - have fun"
ElseIf (dayNum>=2 AND dayNum<=6)
   Print "Weekday - go to work"
Else
   Print "Only 1 through 7"
End If
```

Case Stracame version

```
'assign dayNum or read in value
Select Case (dayNum)
   Case 1, 6: Print "Weekend - have fun"
   Case 2 To 7: Print "Weekday - go to work"
                        'the word "To" means all values 2 through 7
   Case Else: Print "Only 1 through 7"
End Select
```


4.5 What is the output of this section of code?

```
temp=80;
if (temp>80)
   if (temp>90)
      cout<<"Hot";
else
   cout<<"Warm";
cout<<"day"<<endl;
```

OUTPUT: day

Because temp is not greater than 80, the first condition is false, and the second if..else is never executed. The else always goes with the closest if, even if the indentation is not correct. The only time "Warm day" would be the output would be for temp values of 81 through 90.

4.6 Write a Visual Basic code section to count down from 10 to 0 and then print "Blast off!"

```
For lcv=10 To 0 Step -1   'in Visual Basic the For statement goes from the initial
                          'value to the final value, incrementing
   Print lcv              'according to the Step value. If the Step is omitted, the
                          'default is to add 1.
Next lcv
Print "Blast off!"
```

4.7 Write a Visual Basic code section to calculate the value of $1,000 at the end of 5 years, with a simple interest rate of 5 % per year.

```
rate=0.05
amt=1000
For year=1 To 5   'year gets values from 1 to 5, with a default increment of 1
   amt=amt+amt*rate
   Print amt
Next year
```

4.8 What is the output of this section of code if the sales amounts are 10, 13, 15, and 20?

```
int row, col, sales;
char response;
cout<<'Do you want to see a graph of your sales?';
cin>>response;
while (response=='y' || response=='Y')
{
```

OUTPUT:

Loop structures can be nested within other structures. The pretest loop in this example continues as long as the user replies with a Y or y to the question. The two inner loops print the rows and columns of the histogram.

4.9 Write two versions of Visual Basic code to implement the menu in Example 4.20.

```
Do
   Print "1. Add"
   Print "2. Subtract"
   Print "3. Multiply"
   Print "4. Divide"
   Print "5. Quit"
   choice=InputBox("Enter your choice")
Loop While (choice<>5)
```

The "not equal to" symbol in Visual Basic is "<>." This menu always prints once, and continues until the user enters 5.

"Until" version

```
Do
    Print "1. Add"
    Print "2. Subtract"
    Print "3. Multiply"
    Print "4. Divide"
    Print "5. Quit"
    choice=InputBox("Enter your choice")
Loop Until (choice=5)
```

4.10 Write a short C++ program which will implement the simple calculator program from Solved Problem 2.3. Use the pseudocode as comments to guide the code-writing process.

Answer.

```
#include <iostream.h>

void main ()
{
   //declare variables
   int num1,num2,result;
   char oper;

   //  1. Get two numbers and the operation desired
   cout<<endl<<"Enter number one=>"; cin>>num1;
   cout<<endl<<"Enter number two=>"; cin>>num2;
   cout<<endl<<"Enter operation=>"; cin>>oper;

   //  2. Check the operation and perform the correct operation
   switch (oper)
   {
      case '+': result=num1+num2; break;
      case '-': result=num1-num2; break;
      case '*': result=num1*num2; break;
      case '/': if (num2==0)
                        {
                           result=0;
                           cerr<<"Number 2 cannot be a zero"<<endl;
                        }
                        else result=num1/num2;
                        break;
      default:
                        cerr<<"Operation must be + - * or /"<<endl;
   }//end switch
   // 3. Print out the result or the error message
   cout<<endl<<"The result of" <<num1<< oper<<num2<<" is "<<result<<endl;
}
```

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

4.1 Tìm giá trị của mỗi biểu thức sau đây ứng với các giá trị a, b và c của biến Boole đã được cho sẵn. Hãy nhớ tuân thủ thứ tự ưu tiên trong bảng 4.5.

4.2 Tìm giá trị của mỗi trong biểu thức sau đây ứng với các giá trị a, b và c được co và biến Boolean được thực hiện? Hãy tuân thủ thứ tự ưu tiên trong bảng 4.5.

4.3 Hãy viết một câu lệnh quan hệ hoặc lệnh logic trong Visual Basic và C và C++ sẽ trình bày đoạn sau đây:

4.4 Hãy viết mã Visual Basic dành cho ba kiểu cấu trúc chọn lựa trong ví dụ 4.11.

4.5 Tìm kết quả xuất của đoạn mã sau đây?

4.6 Hãy viết một đoạn mã Visual Basic để đếm xuống từ 10 cho đến 0 rồi in "Blast off".

4.7 Hãy viết một đoạn mã Visual Basic để tính giá trị của 1.000 đô vào cuối 5 năm với tỷ lệ lãi suất đơn là 5% trong một năm.

4.8 Tìm kết quả xuất của đoạn mã nếu lượng doanh số là 10, 13, 15 và 20.

4.9 Hãy viết hai biên bản của mã Visual Basic để thực thi menu trong ví dụ 4.20.

4.10 Hãy viết một chương trình ngắn C++ sẽ thực thi chương trình của máy tính đơn giản từ bài tập có lời giải 2.3. Sử dụng mã giả là các phần bình chú để hướng dẫn quy trình viết mã.

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

For all these problems, assume the variables have all been declared properly.

4.11 Write a relational or logical statement in Visual Basic and C/C++ that will express the following:

Expression
(a) distance is 200 miles or greater
(b) number is evenly divisible by 4
(c) x equals y or y equals z but z does not equal x
(d) both x and y are positive
(e) x is negative or y is positive, but not both

4.12 Write a section of code (in both Visual Basic and C++) to determine whether a year is a leap year. A leap year is any year divisible by 4 unless it is divisible by 100, but not 400.

4.13 Write a section of code (in both Visual Basic and C++) that will print all the perfect squares between 0 and 100.

4.14 Write a section of code in C++ that will print the following designs:

(a)
```
*
**
***
****
*****
```

(b)
```
*****
****
***
**
*
```

4.15 Repeat the previous exercise using Visual Basic.

4.16 Write a segment of C++ code that will get an integer as input, and then use a posttest loop to print a list of powers of that integer untill the power is greater than 10,000.

4.17 Write a segment of Visual Basic code that will get an integer as input, and then use a posttest loop to print the square, then the

square of the square, and continue with successive squares until the square is greater than 100,000.

418 Indicate what the output will be for the following Visual Basic segments of code:

(*a*)
```
num1=0
num2=10
Do
    num1=num1+1
    num2=num2-1
    Print num1, num2
Loop While num1<num2
```

(*b*)
```
num=1
Do
    Print num, 17 Mod num
    num=num+1
Loop While 17 Mod num<>5
```

(*c*)
```
num1=4
num2=80
Do
    num2=num2/num1-6
    If num2>num1 Then
        num2=num1+20
    End If
Loop While num2>=0
Print num1, num2
```


4.19 Indicate what the output will be for the following C++ segments of code:

(*a*)
```
num=15;
do
{
    cout<<num<<''<<num/3<<endl; //integer division
    num--;
}while (num/3>3);
```

(*b*)
```
num1=2;
do
{
    cout<<num1<<'';
    num1*=2;
} while (num1<=20);
```

(c)
```
sum=0;
num1=7;
while (num1<10)
{
    for (num2=num1; num2<=10; num2++)
        sum+=num2;
    num1++;
}
cout<<sum<<endl;
```

4.20 Write the VB program which will implement the hotel checkout program from Solved Problem 2.5. Assume the meal charge is $9.95 per person per night, and the tax rate is 5%.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

Đối với những bài tập này, ta giả sử các biến đều được khai báo hoàn chỉnh.

4.11 Hãy viết một câu lệnh quan hệ hoặc logic trong Visual Basic và C/ C++ để trình bày nội dung sau đây

4.12 Hãy viết một đoạn mã (cả trong Visual Basic và C++) để xác định xem thử một năm có phải là năm nhuận hay không. Một năm nhuận là một năm bất kỳ mà chia hết cho 4 trừ khi nó chia hết cho 100 nhưng không chia hết cho 400.

4.13 Hãy viết một đoạn mã (cả trong Visual Basic và C++) để in tất cả các bình phương hoàn chỉnh giữa 0 và 100.

4.14 Hãy viết một đoạn mã trong C++ để in các dấu sau đây:

4.15 Lặp lại bài tập trước đây bằng cách sử dụng Visual Basic.

4.16 Hãy viết một đoạn mã C++ để nhận một số nguyên làm đối tượng nhập rồi sử dụng phép lặp thử nghiệm sau để in một danh sách lũy thừa của số nguyên đó cho đến khi lũy thừa này lớnhơn 10.000.

4.17 Hãy viết một đoạn mã Visual Basic sẽ nhận một số nguyên làm giá trị nhập, rồi sử dụng một phép lặp thử nghiệm sau để in bình phương, sau đó bình phương của bình phương tiếp tục với các bình phương tuần tự cho đến khi bình phương kết quả lớn hơn 100.000.

4.18 Chỉ định cho biết đối tượng xuất sẽ như thế nào đối với đoạn mã Visual Basic sau đây.

4.19 Hãy chỉ định cho biết kết quả xuất sẽ như thế nào ứng với đoạn mã C++ sau đây.

4.20 Hãy viết một chương trình VB sẽ thực thi chương trình kiểm toán khách sạn từ bài tập 2.5. Giả sử rằng chi phí bữa ăn là 9.95 tính trên mỗi người trong một đêm và mức thuế là 5%.

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

4.11 Solution:

	C/C++	VB
(*a*)	(distance>= 200)	(distance>= 200)
(*b*)	(number % 4==0)	(number MOD 4 = 0)
(*c*)	((x==y || y==z) && (y!= z))	((x = y OR y = z) AND (y<>z))
(*d*)	(x>=0 && y>=0)	(x>=0 AND y>=0)
(*e*)	((x<0 && y<0) || (x >=0 && y>=0))	((x<0 AND y<0) OR (x >=0 AND y>=0))

4.12 C++ version:

```
cout<<'Enter the year'; cin >>year;
if (year%4==0)
{
    if (year%100!=0)
       cout<<year<<'is a leap year'<<endl;
    else if (year%400==0)
       cout<<year<<'is a leap year'<<endl;
    else
       cout<<year<<'is NOT a leap year'<<endl;
    }
    else
       cout<<year<<'is NOT a leap year'<<endl;
```

Visual Basic version:

```
year=Val(txtYear)
If (year Mod 4=0) Then
    If (year Mod 100 <> 0) Then
       Print year; 'is a leap year'
    ElseIf (year Mod 400=0) Then
       Print year; 'is a leap year'
    Else
       Print year; 'is NOT a leap year'
    End If
Else
  Print year; 'is NOT a leap year'
End If
```

4.13 C++ version:

```
num=0;
numSqr=num*num;
while (numSqr<100)
{
   cout<<numSqr<<' ';
   num++;
   numSqr=num*num;
}
```

Visual Basic version:

```
Do While (numSquare<100)
   Print numSquare
   num=num+1
   numSquare=num*num
Loop
```

4.14 (*a*)
```
for (row=1; row<=5; row++)
{
   for (col=0; col<row; col++)
      cout<<'*';
   cout<<endl;
}
```

(*b*)
```
for (row=1; row<=5; row++)
{
   for (col=6-row; col>0; col--)
      cout<<'*';
   cout<<endl;
}
```

4.15 (*a*)
```
For row=1 To 5
   For col=1 To row
      Print "*";
   Next col
   Print
Next row
```

(*b*)
```
For row=1 To 5
   For col=6-row To 1 Step -1
      Print "*";
   Next col
   Print
Next row
```

4.16
```
power=1;
cout<<'Enter an integer greater than 1=>';
cin>>num;
do
{
    cout<<power<<'';
    power *=num;
}while (power<10000);
```

4.17
```
num=InputBox('Enter an integer greater than 1=>')
num=num*num
Do
    Print num
    num=num*num
Loop While (num<100000)
```

4.18 Output (*a*)

1 9
2 8
3 7
4 6
5 5

Output (*b*)

1 0
2 1
3 2
4 1
5 2

Output (*c*): 4 −6

4.19 Output (*a*):

15 5
14 4
13 4
12 4

Output (*b*): 2 4 8 16

Output (*c*): 80

4.20 VB program:

```
Sub main()
  'Declare variables
  Const MEALCHARGE=9.95
  Const ROOM1PERSON=55
  Const ROOM2PEOPLE=60
  Const ROOM3ORMORE=65
  Const TAXRATE=0.05
  Dim nights As Integer, people As Integer
  Dim meals As Single, total As Single
  '1. Get the number of nights and number of people
  'in the room from the customer
  nights=InputBox("How many nights?")
  people=InputBox("How many people?")
  '3. Calculate bill
  '3.1. Look up the room charge for that number of people
  '3.2. Multiply by the number of nights
  If (people=1) Then
    total=nights*ROOM1PERSON
  ElseIf (people=2) Then
    total=nights*ROOM2PEOPLE
  ElseIf (people>2) Then
    total=nights*ROOM3ORMORE
  Else 'if number is zero or negative
    total=0
  End If
  '3.3. Add the meal charges
  total=total+(people*MEALCHARGE)
  '3.4. Calculate and add the tax
  total=total+(total*TAXRATE)
  '4. Print the bill
  MsgBox ("Your total cost is" +Str(total))
End Sub
```

CHAPTER 5

Functions and Subroutines

Các hàm và các thường trình con

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các khái niệm về hàm, các thường trình con và các cơ chế truyền tham số khi viết chương trình, cụ thể với các nội dung cơ bản sau đây:

- Functions — *Các hàm*
- Subroutines — *Các thường trình con*
- Scope and lifetime of identifiers — *Phạm vi và thời gian tồn tại của các bộ nhận dạng*
- Parameter-passing mechanisms — *Cơ chế truyền tham số*

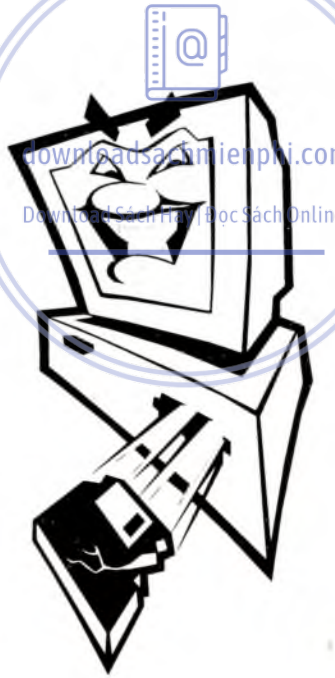
Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.

GIỚI THIỆU

In any software development process, subroutines and functions play a critical role since they allow the decomposition of the program into logical units or modules. A **subroutine** is a self-contained program structure included within a program. Subroutines are written to solve a particular problem; they have a specific purpose. **Functions, like** subroutines, are also self-contained program structures designed with a single purpose; however, unlike subroutines they all "return" a single value of a particular type. This unique feature allows programmers to use a function in the same context where a variable or expression can be used. The term **subprogram** is commonly used to refer to either a function or a subroutine.

Before concentrating our attention to the specific aspects of functions and subroutines, let us consider how they facilitate program planning. **Modular programming** is the application to computer programming of the principle "divide and conquer" used so successfully by other disciplines to solve complex problems. What needs to be "conquered" (or solved) is a

particular problem and, to solve it, we divide it into smaller and more manageable subproblems. Each of these subproblems can be, in turn, subdivided even further until they are considered by the programmer to be "elementary" or "sufficiently simple." Since the notion of "elementary" or "simple" is a subjective notion, we will adopt the idea that a task is "elementary" when its function can be described by a sentence that contains a single subject, a single verb, and a single object. For example, see Solved Problems 5.1 and 5.2. As self-contained structures, subroutines and functions can be used as building blocks to construct a program. In this sense, we can use the same routine or function in different programs whenever is appropriate. We can also replace less efficient program components by faster or simpler ones without affecting the execution of the program.



CHỦ ĐIỂM 5.1

FUNCTIONS
Các hàm

As indicated before, a function is a program structure that always returns a single value. A function may receive one or more input values. These input values are called **arguments or parameters** (see Fig. 5-1).

Functions may be "called" or "invoked" from a main program, function, or subroutine. Functions are called by naming them and passing the appropriate parameters, if any. Whenever a function is called the execution of the main program, subroutine, or function that calls the function is halted. The control of the program is automatically transferred to the function which, upon completion, returns to the same statement where it was called (its reference point). Functions can be used whenever a variable can be used.

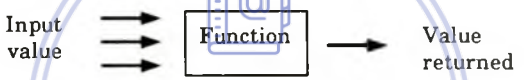


Fig. 5-1 General structure of a function.

Some of the languages that we use to illustrate the examples in this book require that all functions be "declared" before they get called (see Solved Problem 5.3). This means that we need to specify the function's name, the data type that it returns, the number of arguments or parameters that it receives, if any, the order of these parameters, and their data type. The term argument(s) or parameter(s) refers to the input that the function receives.

Although the syntax rules for declaring functions may vary from language to language, in any function we can always distinguish two basic elements. Using the terminology of the American National Standards Institute (ANSI) we will call these basic elements the **function header** and the **function body** (see Fig. 5-2). The header provides general information about the function's name, the type that it returns, the number of input parameters, if any, the order of these parameters, and their types. Depending on the language, the syntax of the function header may require the use of one or more keywords and some other qualifiers (see Example 5.1). The body is made up of local declarations and some other statements that perform the function's task.

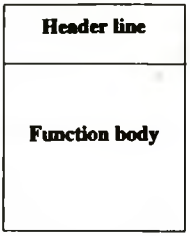


Fig. 5-2 Basic elements of a function

EXAMPLE 5.1 Distinguish the basic elements of the C/C++ function shown below

```
int max_of_three_integers (int a, int b, int c)
{ int maximum;

    maximum = a;
    if (b > maximum)
       maximum = b;
    else
    if ( c > maximum)
        maximum = c;
    return (maximum); }
```

Header

body

The header of this function is: int max_of_three_integers (int a, int b, int c).

This header indicates:

(1) the name of the function (max_of_three_integers)

(2) the data type that it returns (int)

(3)- the number of inputs of the function (int a, int b, int c). This function receives three input integer parameters called a, b, and c, respectively. The variables a, b, c, and their data type are called the formal **parameters** or **arguments** of the function.

The body, enclosed in braces or curly brackets, consists of:

```
{ int maximum;
  maximum=a;
  if (b>maximum)
        maximum=b;
  else
```

```
    if (c>maximum)
          maximum=c;
  return (maximum); }
```

(1) a local declaration of the integer variable called maximum.

(2) the instructions that calculate the maximum of the three given integer numbers. In C, the variable or expression indicated in parentheses in the return statement is evaluated and "sent back" to its referencing point in the calling procedure. The term "calling procure" refers to the main program, subroutine, or function that "calls" the function. The following example illustrates this.

EXAMPLE 5.2 Identify the calling program, the calling statement and explain the flow of control in the following program section of a C++ program.

```
int main()
{ int val1,val2,val3, max;
  cin>>val1>>val2>>val3; //input the three integers
  max=max_of_three_integers (val1,val2, val3) //this statement calls the function
  cout<<"The maximum of the three integers is" <<max; //print the maximum
  return (0); }
int max_of_three_integers (int a, int b, int c)
{ int maximum;
  maximum=a;
  if (b>maximum)
    maximum=b;
  else
    if (c>maximum)
      maximum=c;
return (maximum);}
```


The main program (the calling program), after receiving three input values from the user, calls the function max_of_three_integers(val1,val2,val3) to calculate the maximum value of the three integers. The execution of the main program is halted as soon as it reaches the statement that contains the function call. At this moment, the function starts executing. Using computer lingo, we can say that "control is transferred to the function." After the function calculates the maximum value, the function returns to the calling statement (its reference point) and the returned value is assigned to the variable max. The main program then continues executing its remaining statements.

In Example 5.2, notice that when the function gets called the parameter values val1, val2, and val3 are separated by commas and enclosed in parentheses following the name of the function. These values are the input values that the function receives. We say that these values are "passed" to

the function and we will call them the **actual parameters.** As **Fig.** 5-3 shows, the actual parameters are associated with the formal parameters "by position." That is, the value of actual parameter, val1, is assigned to the formal parameter, a, of the function. Likewise, the values of variables val2 and val3 are assigned to the formal parameters b and c respectively.

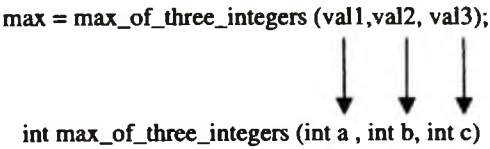


Fig. 5-3 Association of actual and formal parameters by position.

Example 5.2 also illustrates the rules that govern the calling of a function. These rules, which apply to any program, subroutine, or function that calls another function, are:

(1) The number of actual parameters in the calling statement must be the same as the number of formal parameters in the header of the function. There are languages like Visual Basic where a parameter can be defined as optional. However, we will not consider this type of parameter in this book.

(2) The type of the actual parameters must be of the same type as its corresponding formal parameters.

(3) Actual parameters are generally associated to formal parameters "by position." Depending on the language, the association may be carried out by pairing the actual and formal parameters from left to right or from right to left or in no particular order.

Visual Basic allows a different mechanism for passing multiple arguments, where the parameters are explicitly named in the call to the function. This new modality eliminates the need for passing arguments according to the order specified in the function header. This feature does not affect the syntax or structure of the function header at all. The following example illustrates this.

EXAMPLE 5.3 Assume that we have the following function declaration which accepts three input parameters and an instance of a call to this function:

Private Function IsAlarmClockSet (iSecs As Integer, iMins As Integer, iHrs As Integer) As Boolean

An instance of a call to this function may be IsAlarmClockSet (30, 45, 13).

Notice that whenever we call this function, we need to remember what these parameters are and what they represent. However, if we use named arguments this is not necessary.

To call the previous function using named arguments, each formal parameter is associated with its formal parameter using the following syntax:

FunctionName(formalParl : = actualParl, formalPar2' : = actualPar2, formalParN : = actualParN)

Using this mechanism, we could call the function as follows:

bVariable = IsAlarmClockSet(iMins : = 45, iHrs : = 13, iSecs : = 30)

or

bVariable = IsAlarmClockSet(iHrs : = 13, iSecs : = 30, imins : = 45)

Notice that the syntax of the call to the function requires the use of the operator : = to associate a formal parameter with its actual parameter.

In some languages like C and C++ it is required that the function be declared using a **function prototype.** In these languages the prototype gives the name of the function, the value that it returns and the number and type of the formal parameters. The function prototype helps the compiler to do error checking (see Appendix A). A compilation error occurs if the information provided by the prototype does not agree with that of the function header or with the type of the value returned by the function. If a compilation error of this type occurs the program does not get executed. A prototype for the function of Examples 5.1 and 5.2 may look like this:

```
int max_of_three_integers (int, int, int);
```

In this function prototype there is no mention of the name of the actual parameters, just their type. However, it is also possible to name the arguments as part of the prototype, as indicated below:

```
int max_of_three_integers (int x, int y, int z);
```

In this case the variables x, y, and z are "dummy" arguments. These variables do not need to appear in the function header. We can declare the function used in the two previous examples with this prototype. No changes need to be made either to the calling program or the function itself.

The value that a function returns can be omitted in some languages. In all these cases a predefined value is used as default. In the case of the C language the default value is int. Since this rule may vary from language to language, it is always wise to consult the reference manual of a language to find out whether the default value can be used or not.

The proper place of the function prototype inside the program will be discussed later in this chapter.

In C and C++ the value that the function returns is explicitly indicated by means of a return statement. However, not all languages use this mechanism. The following example shows the implementation of the max_of_three_integers function using Visual Basic.

EXAMPLE 5.4 Identify the basic elements of the following Visual Basic function.

```
Private Function MaxOfThreeIntegers (a As Integer, b As Integer, c As Integer) As Integer
Dim maximum As Integer 'declaration of variable maximum as an integer
maximum=a
If b>maximum Then maximum=b
Elseif c>maximum Then maximum=c
End if
MaxOfThreeIntegers=maximum 'Notice that the function name is treated as a variable
End Function
```


The header of this function is Private Function UvIax0f1breeIntegers (a As Integer, b As Integer, c As Integer) As Integer

The header is comprised of several parts and indicates:

(1) the scope or visibility of the function (see Section 5.3). The word private signals that this function is known only within the form or module in which it is defined.

(2) the name of the function. Notice that we have written the name of the function as a combination of upper- and lowercase letters with the letter i as prefix. The letter i is a mnemonic to the programmer or any other reader that the function returns an integer value. This style of writing names is known as the Hungarian notation. The name of the function must be followed by parentheses even if there are no parameters.

(3) the data type that the function returns. The As Integer following the formal parameters states explicitly that this particular function returns an integer.

(4) the number and type of the formal parameters. This function has three input integer parameters a, b, and c. The As Integer allows us to define the data type of the formal parameter.

The body of this function follows the header and ends with the **words End Function.**

The code to calculate the maximum value is almost identical to the code of C and C++ shown before. However, notice that in Visual Basic we must specify the function's return value within the function's body. We do this by treating the function's name as if it were a variable and assigning a value to it.

One interesting exception to a function returning a value is observed in C and C++ with the use of the keyword **void.** This word, when placed before a function's name, indicates that the function will return no value. Although this may seem to be a contradiction of the definition of a function, it is useful in those situations where the programmer is more interested in the effect of the function rather than the value that it returns.

CHÚ THÍCH TỪ VỰNG

subroutine:	*thường trình con*
Function:	*hàm*
subprogram:	*chương trình con*
Modular programming:	*lập trình mô đun*
arguments:	*các đối số*
parameters:	*các tham số*
function header:	*đầu đề hàm*
function body:	*thân hàm*
actual parameters :	*các tham số thật sự*
function prototype:	*kiểu mẫu của hàm*

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 5.1

Với quy trình phát triển phần mềm nào, các trình con và các hàm đóng một vai trò chuẩn mực bởi vì chúng cho phép phân rã chương trình thành các đơn vị hợp lý hoặc các modun hợp lý. Một trình con tự bản thân là một cấu trúc chương trình được đưa vào trong một chương trình khác. Các trình con thường được viết để giải quyết một bài toán đặc biệt; có một mục tiêu đặc biệt. Các hàm cũng giống như các trình con, tự bản thân là các cấu trúc chương trình được thiết kế với một mục đích; tuy nhiên không giống như trình con, tất cả chúng đều cho ra một giá trị đơn của một loại đặc biệt. Tính năng đồng nhất này cho phép nhà lập trình sử dụng hàm trong ngữ cảnh giống hệt như ngữ cảnh mà một biến hoặc một biểu thức có thể được dùng. Thuật ngữ chương trình con thường được dùng phổ biến để ám chỉ hoặc một hàm hoặc một trình con.

Trước khi tập trung chú ý vào khía cạnh đặc biệt của các hàm và thường trình con, chúng ta hãy xem xét cách mà chúng đặt vai trò trong việc quy hoạch chương trình. Lập trình modun là trình ứng dụng cho lập trình máy tính dựa theo nguyên lý "phân chia và giải quyết" được dùng một cách thành công do bởi các nhà nghiên cứu để giải quyết các bài toán phức tạp. Những gì mà chúng ta cần phải "giải quyết" chỉ là một bài toán đặc biệt và để giải nó chúng ta chia nó thành các bài toán nhỏ hơn có thể quản lý được. Mỗi một bài toán nhỏ này sau đó lần lượt được chia nhỏ hơn nữa cho đến khi chúng được nhà lập trình xem như là yếu tố "sơ cấp" hoặc đủ "đơn giản". Bởi vì yếu tố "sơ cấp"hoặc "đơn giản" là một ghi chú về chủ điểm nên chúng ta cần mô phỏng ý tưởng cho rằng một tác vụ là "yếu tố nguyên thủy, yếu tố sơ cấp" lúc chức năng của nó có thể được mô tả bởi câu có chứa một chủ từ, một động từ và một túc từ. Ví dụ, xem bài tập có lời giải 5.1 và 5.2. Dưới dạng là cấu trúc tự chứa, các trình con và các hàm có thể được dùng làm cấu trúc khối để xây dựng một chương trình. Theo ý nghĩa này, thì chúng ta có thể sử dụng cùng một trình con hoặc một hàm trong nhiều chương trình khác nhau bất cứ lúc nào phù hợp. Chúng ta cũng có thể thay thế các thành phần chương trình ít hiệu quả bằng các chương trình nhanh hơn hoặc đơn giản mà không ảnh hưởng đến việc thực thi chương trình đó.

5.1 CÁC HÀM

Như đã đề cập trước đây, một hàm chính là một cấu trúc chương trình luôn luôn trả về một giá trị đơn. Một hàm có thể nhận một hoặc nhiều giá trị đầu vào. Những giá trị đầu vào này được gọi là các đối số hoặc tham số (xem hình 5.1).

Các hàm có thể được "gọi" hoặc "viện dẫn" từ một chương trình chính, một hàm, hoặc một trình con. Các hàm còn được gọi theo tên của chúng bằng cách chuyển các tham số phù hợp nếu có. Bất cứ lúc nào một hàm được gọi thì việc thực thi chương trình chính, trình con hoặc hàm vốn gọi hàm đó đều bị ngưng. Phần điều khiển chương trình tự động được truyền sang hàm thực thi để trả về câu lệnh giống như câu lệnh đã được gọi (điểm tham chiếu của nó). Các hàm có thể

***Hình 5.1** Cấu trúc thông thường của một hàm*

Một vài ngôn ngữ mà chúng ta sử dụng để minh họa ví dụ trong sách này yêu cầu rằng tất cả các hàm phải được "khai báo" khi chúng được gọi (xem bài tập có lời giải 5.3). Điều này có nghĩa rằng chúng ta phải

chỉ tên của hàm, kiểu dữ liệu trả về, số của các đối số hoặc tham số có nhận nếu có, thứ tự của các tham số này, và kiểu dữ liệu của chúng. Thuật ngữ đối số hoặc tham số ám chỉ đến đầu vào mà hàm nhận được.

Mặc dù quy tắc cú pháp dùng để khai báo các hàm có thể thay đổi tùy theo từng ngôn ngữ, nhưng trong bất cứ hàm nào chúng ta đều luôn luôn phân biệt hai yếu tố căn bản. Bằng cách sử dụng thuật ngữ của viện tiêu chuẩn Hoa Kỳ (ANSI) chúng ta sẽ gọi các yếu tố căn bản này là tiêu đề hàm (function header) và nội dung hàm (function body) (xem hình 5.2). Tiêu đề cung cấp thông tin tổng quát về tên của hàm, kiểu mà nó trả về, số các tham số đầu vào nếu có, thứ tự của các tham số này và kiểu. Phụ thuộc vào ngôn ngữ, cú pháp của tiêu đề hàm yêu cầu phải sử dụng một hoặc nhiều từ khóa và một vài lượng từ khác (xem ví dụ 5.1). Bố cục tạo nên các khai báo cục bộ và một vài câu lệnh khác để thực thi tác vụ của hàm.

***Hình 5.2** Các yếu tố căn bản của một hàm*

***VÍ DỤ 5.1** Phân biệt các yếu tố căn bản của hàm C/C ++ được minh họa dưới đây.*

```
int max_of_three_integers (int a, int b, int c)
{ int maximum;

    maximum = a;
    if (b > maximum)
       maximum = b;
    else
    if ( c > maximum)
        maximum = c;
    return (maximum); }
```

Tiêu đề

nội dung

Tiêu đề của hàm này là: int max_of_three_integers (int a, int b, int c).

Tiêu đề hàm chỉ định những vấn đề sau đây:

(1) tên của hàm (max_of_three_integers)

(2) kiểu dữ liệu mà nó trả về (int)

(3) số các đầu vào của hàm (int a, int b, int c). Hàm này nhận ba tham số nguyên đầu vào có tên là a, b và c tương ứng. Các biến a, b và c và kiểu dữ liệu của chúng được gọi là các tham số chính hoặc các đối số của hàm.

Nội dung, được đặt bên trong các dấu ngoặc đơn hoặc dấu móc, bao gồm"

```
{ int maximum;
  maximum=a;
  if (b>maximum)
        maximum=b;
  else
  if (c>maximum)
        maximum=c;
return (maximum); }
```

(1) phần khai báo cục bộ của biến nguyên được gọi là maximum.

(2) các câu lệnh để tính toán giá trị cực đại của các số nguyên đã cho. Trong C các biến hoặc biểu thức được chỉ định trong các dấu móc đơn trong câu lệnh return được lượng giá và "gởi trở lại" điểm tham chiếu của nó trong thủ tục gọi. Thuật ngữ "thủ tục gọi" chỉ đến chương trình chính, trình con hoặc hàm vốn "gọi" hàm này. Ví dụ sau đây minh họa điều này.

***VÍ DỤ 5.2** Nhận biết chương trình gọi, câu lệnh gọi và giải thích dòng điều khiển trong một chương trình sau đây của chương trình C++.*

```
int main()
{ int val1,val2,val3, max;
  cin>>val1>>val2>>val3; //input the three integers
  max=max_of_three_integers (val1,val2, val3) //this statement calls the function
  cout<<'The maximum of the three integers is' <<max; //print the maximum
  return (0); }
int max_of_three_integers (int a, int b, int c)
{ int maximum;
  maximum=a;
  if (b>maximum)
    maximum=b;
  else
    if (c>maximum)
      maximum=c;
return (maximum);}
```

Chương trình chính (chương trình gọi), sau khi nhận ba giá trị đầu vào từ người dùng, thì sẽ gọi hàm max_of_three_integers(val1, val2, val3) để tính giá trị cực đại của ba số nguyên này. Việc thực thi chương trình chính được tạm ngưng khi nó đạt đến câu lệnh có chứa hàm gọi. Vào lúc này, thì hàm bắt đầu thực thi. Sử dụng ngôn ngữ máy tính, chúng ta có thể nói rằng "chức năng kiểm soát được truyền vào hàm". Sau khi hàm tính giá trị cực đại, hàm trả về câu lệnh gọi (điểm tham chiếu của nó) và giá trị trả về sẽ được gán vào variable max (max biến). Chương trình chính sau đó tiếp tục thực thi các câu lệnh còn lại.

Trong ví dụ 5.2, cần lưu ý rằng lúc hàm nhận các giá trị tham số được gọi val1, val2, và val3 tách rời nhau bởi dấu phẩy và được đặt trong các dấu móc đơn theo sau tên của hàm. Những giá trị này chính là những giá trị nhập mà hàm nhận. Chúng ta bảo rằng những giá trị này được "chuyển đến" hàm và chúng ta sẽ gọi chúng là các tham số thật sự. Như minh họa trong hình 5.3, các tham số thật sự được liên kết với các tham số hình thức "bởi vị trí". Có nghĩa rằng giá trị của tham số thật sự , val1 được gán với tham số hình thức a của hàm. Tương tự như vậy giá trị của tham số val2 và val3 được gán với các tham số hình thức b và c tương ứng.

```
max = max_of_three_integers (val1, val2, val3);
```

```
int max_of_three_integers (int a , int b, int c)
```


Hình 5.3 *Liên kết của tham số thật sự và tham số hình thức bằng vị trí.*

Ví dụ 5.2 Cũng minh họa các quy tắc điều phối cuộc gọi một hàm. Những quy tắc này áp dụng cho bất cứ chương trình, trình con hoặc hàm nào để gọi hàm khác, đó là:

(1) Số các tham số thực tế trong câu lệnh gọi phải bằng số các tham số hình thức trong tiêu đề của hàm. Có những ngôn ngữ như Visual Basic ở đó một tham số có thể được xác định tùy ý. Tuy nhiên, chúng ta sẽ không xem xét kiểu tham số đó trong sách này.

(2) Kiểu các tham số thực tế phải giống hệt như kiểu các tham số hình thức tương ứng của nó.

(3) Các tham số thực tế thường được liên kết với các tham số hình thức "qua vị trí". Phụ thuộc vào ngôn ngữ, sự liên kết này có thể được thực thi bằng cách kết cặp các tham số thực tế và hình thức từ trái sang phải hoặc từ phải sang trái hoặc không theo một thứ tự đặc biệt nào cả.

Visual Basic cho phép một cơ cấu khác để truyền qua nhiều tham số ở đó các tham số được đặt tên minh nhiên trong cuộc gọi đến hàm này. Hình thức mới này loại bỏ nhu cầu truyền các đối số theo thứ tự được chỉ định trong tiêu đề hàm. Tính năng này không tác động đến cú pháp cấu trúc của tiêu đề hàm chút nào cả. Ví dụ sau đây minh họa điều này.

VÍ DỤ 5.3 *Giả sử rằng chúng ta có khai báo hàm sau đây vốn chấp nhận ba tham số vào và một hằng số gọi cho hàm này:*

```
Private Function IsAlarmClockSet (iSecs As Integer, iMins As Integer, iHrs As
Integer) As Boolean
```

Một hằng số gọi cho hàm này có thể là IsAlarmClockSet (30, 45, 13).

Lưu ý rằng bất cứ lúc nào chúng ta gọi hàm này, chúng ta cần nhớ bản chất của những tham số này là gì và chúng tiêu biểu cho điều gì. Tuy nhiên, nếu chúng ta sử dụng các đối số đã có tên thì điều này không cần thiết.

Để gọi hàm trước đây bằng cách sử dụng các đối số có tên, mỗi tham số hình thức được liên kết với tham số hình thức của nó bằng cách sử dụng cú pháp sau đây:

```
FunctionName(formalPar1: = actualPar1, formalPar2: = actualPar2:=
actualPar2, formalParN: = actualParN)
```

Bằng cách sử dụng cơ cấu này chúng ta có thể gọi hàm như sau:bVariable = IsAlarmClockSet(iMins: = 45, iHrs: = 13, iSecs: = 30)

hoặc

```
bVariable = IsAlarmClockSet(iHrs: = 13, iSecs:= 30, iMins: = 45)
```

Lưu ý rằng cú pháp của cuộc gọi đến hàm yêu cầu phải sử dụng toán tử := để liên kết một tham số hình thức với tham số thực tế.

Trong một vài ngôn ngữ như C và C++ người ta yêu cầu rằng hàm phải được khai báo bằng cách sử dụng hình thái hàm (function prototype). Trong ngôn ngữ này, hình thái cho ta tên của hàm, giá trị mà nó trả về số và kiểu tham số hình thức. Hình thái hàm giúp cho người biên soạn kiểm tra lỗi (xem phụ lục A). Một lỗi biên soạn xảy ra nếu thông tin do hình thái cung cấp không phù hợp với thông tin của tiêu đề hàm hoặc kiểu giá trị mà hàm trả về. Nếu có một lỗi biên soạn của kiểu này xảy ra thì chương trình không được thực thi. Một hình thái dùng cho hàm trong ví dụ 5.1 và 5.2 giống như dưới đây:

```
int max_of_three_integers (int, int, int);
```

Trong hình thái hàm này không có sự đề cập nào đến tên của các tham số thực tế, chỉ có kiểu mà thôi. Tuy nhiên, ta cũng có thể đặt tên của tham số như là một phần của hình thái như chỉ định dưới đây:

int max_of_three_integers (int x, int y, int z);

Trong trường hợp này các biến x, y và z là các đối số "hiểu ngầm", thì những biến này không cần phải xuất hiện trong tiêu đề hàm. Chúng ta có thể khai báo hàm được dùng trong hai ví dụ trước đây với kiểu hình thái này. Không có sự thay đổi cần thiết nào xảy ra cho chương trình gọi hoặc cho tự bản thân của hàm. Giá trị một hàm trả về có thể được bỏ qua trong một vài ngôn ngữ. Trong tất cả những trường hợp này, một giá trị được xác định sẵn phải được chọn làm mặc định. Trong trường hợp ngôn ngữ C giá trị mặc định là int. Bởi vì quy tắc này có thể thay đổi tùy theo từng ngôn ngữ, cho nên chúng ta phải tham khảo sổ tay chỉ dẫn của ngôn ngữ để tìm hiểu thêm giá trị mặc định có được dùng hay không được dùng.

Nơi hoàn hảo của hình thái hàm bên trong chương trình sẽ được thảo luận ở phần sau trong chương này.

Trong C và C++ giá trị mà hàm trả về được chỉ định một cách rõ rệt bằng một câu lệnh trả về. Tuy nhiên, không phải tất cả các ngôn ngữ đều sử dụng cơ cấu này. Ví dụ sau đây trình bày việc thực thi hàm max_of_three_integers bằng cách sử dụng Visual basic.

***VÍ DỤ 5.4** Hãy chỉ định các yếu tố căn bản của hàm Visual Basic sau đây.*


```
Private Function MaxOfThreeIntegers (a As Integer, b As Integer, c As Integer) As Integer
Dim maximum As Integer 'declaration of variable maximum as an integer
maximum=a
If b>maximum Then maximum=b
Elseif c>maximum Then maximum=c
End if
MaxOfThreeIntegers=maximum 'Notice that the function name is treated as a variable
End Function
```

Tiêu đề của hàm này là

Private Function MaxOfThreeIntegers(a As Integer, b As Integer, c As Integer) As Integer

Tiêu đề bao gồm nhiều phần chỉ định như sau:

(1) phạm vi hoặc hình dáng của hàm (xem mục 5.3). Từ private cho thấy rằng hàm này chỉ được biết bên trong dạng hoặc module mà nó được xác định.

(2) tên của hàm. Lưu ý rằng chúng ta phải viết tên của hàm dưới dạng

một tổ hợp các mẫu tự chữ hoa và chữ thường với mẫu tự i được chọn làm tiền tố. Mẫu tự i là một thuật nhớ để nhắc nhở nhà lập trình hoặc bất cứ người đọc nào rằng hàm này trả về một giá trị nguyên. Kiểu viết tên được cho dưới hình thức là chú thích Hungari. Tên của hàm phải được đặt trong các dấu móc đơn thậm chí nếu không có tham số.

(3) kiểu dữ liệu mà hàm trả về. As Integer theo sau các tham số hình thức trình bằng một cách minh nhiên rằng hàm đặc biệt này trả về một số nguyên.

(4) số và kiểu của các tham số hình thức. Hàm này có ba tham số nguyên nhập vào a, b và c. As Integer cho phép chúng ta xác định kiểu dữ liệu của tham số hình thức.

Nội dung của hàm này theo sau tiêu đề và kết thúc bằng từ End Function.

Mã để tính giá trị cực đại thì giống hệt như mã của C và C++ được minh họa trước đây. Tuy nhiên, cần lưu ý rằng trong Visual Basic chúng ta phải chỉ định giá trị trả về của hàm bên trong nội dung hàm. Chúng ta thực hiện điều này bằng cách xử lý tên của hàm y hệt như nó là một biến và gán một giá trị cho nó.

Có một ngoại lệ rất là thích thú đối với một hàm trả về một giá trị được xem xét trong C và C++ với cách dùng từ khóa void. Từ này lúc được đặt trước một tên hàm, thì nó cho biết rằng hàm này sẽ không trả về giá trị nào. Mặc dù, điều này dường như mâu thuẫn với định nghĩa của một hàm, nhưng nó sử dụng trong những tình huống ở đó người lập trình quan tâm đến tác động của hàm hơn là giá trị mà nó trả về.

CHỦ ĐIỂM 5.2

SUBROUTINES
Các thường trình con

Subroutines or procedures, as indicated before, are also self-contained program structures. Subroutines, unlike functions, do not have to return a value, and when they do, they use a different mechanism. The declaration of a procedure, for those languages that require it, is similar to that of a function. Likewise, the rules for associating actual and formal parameters are similar to those of a function.

EXAMPLE 5.5 Identify the basic elements in the Visual Basic subroutine shown below.

```
Private Sub EliminateExtraBlanks(txtInputBox As TextBox)
  txtInputBox=RTrim(txtInputBox)
  txtInputBox=LTrim(txtInputBox)
End Sub
```


The header of the subroutine is:

Private Sub EliminateExtraBlanksInTextBox (txtInputBox As TextBox)

The name of the subroutine is EliminateExtraBlanksInTextBox and it receives as an input parameter a TextBox called txtInputBox.

The keyword Private indicates that this subprocedure is known only within the form where it is defined.

The keyword Sub indicates that EliminateExtraBlanks is a subroutine. The body of this subroutine ends with End Sub.

Notice that the body of this subroutine consists of two function calls to the built-in functions Rtrim and Ltrim which eliminate the leading and trailing extra blanks respectively of the input parameter txtInputBox. The actions of these two functions alter the content of the textbox that was passed as an actual parameter (see Section 5.4).

EXAMPLE 5.6 The following C program converts a given Fahrenheit temperature into its Celsius equivalent.

```
#include <stdio.h>
void ConvertToCelsius(void);
double Celsius;
double Fahrenheit;
```

```
void main()
{
  printf ("\nEnter a Fahrenheit temperature :");
  scanf("elf", &Fahrenheit);
  ConvertToCelsius(); printf('\n\n $6.21f Fahrenheit de-
  grees are equivalent to %6.21f Celsius\n\n", Fahrenheit,
  Celsius);
}
void ConvertToCelsius(void)
{
Celsius=(Fahrenheit-32.0)*(5.0/9.0);
}
```

Notice that the "subroutine" ConvertToCelsius has been implemented using a function with void as the returning value. ConvertToCelsius operates on the global variables Celsius and Fahrenheit (see Section 5.3).

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 5.2

5.2 CÁC TRÌNH CON

Các trình con hoặc các thủ tục như đã được chỉ định trước đây là các cấu trúc chương trình tự chứa. Các trình con không giống như các hàm, chúng không trả về một giá trị và lúc chúng thực hiện, thì chúng được sử dụng một cơ cấu khác hẳn. Phần khai báo của một thủ tục dùng cho các ngôn ngữ yêu cầu nó thì giống hệt như phần khai báo của một hàm. Tương tự vậy, các quy tắc để liên kết các tham số thực tế và tham số hình thức thì giống hệt như các quy tắc dùng cho một hàm.

VÍ DỤ 5.5 *Nhận biết các thành phần căn bản trong trình con Visual Basic cho dưới đây.*

```
Private Sub EliminateExtraBlanks(txtInputBox As TextBox)
  txtInputBox=RTrim(txtInputBox)
  txtInputBox=LTrim(txtInputBox)
End Sub
```

Tiêu đề của trình con là

Private Sub EliminateExtraBlanksInTextBox (txtInputBox As textBox)

Tên của trình con là EliminateExtraBlanksInTextBox và nó nhận được một tham số nhập vào một TextBox có tên là txtInputBox.

Từ khóa Private chỉ ra rằng thủ tục con này chỉ được biết bên trong dạng mà nó xác định.

Từ khóa Sub chỉ ra rằng EliminateExtraBlanks là một chương trình con. Nội dung của trình con này kết thúc bằng End Sub.

Lưu ý rằng nội dung của chương trình con này bao gồm hai hàm gọi đến các hàm được tạo sẵn Rtrim và Ltrim để loại bỏ các khoảng trống dư đứng đầu và đứng cuối tương ứng của tham số nhập vào txtInputBox. Các hành động của hai hàm này làm thay đổi nội dung của textbox được truyền qua dưới dạng là một tham số thực tế (xem mục 5.4).

***VÍ DỤ 5.6** Chương trình C sau đây biến đổi một nhiệt độ Fahrenheit sang nhiệt độ tương đương Celsius của nó.*

```
#include <stdio.h>
void ConvertToCelsius(void);
double Celsius;
double Fahrenheit;
void main()
{
  printf ("\nEnter a Fahrenheit temperature : ");
  scanf("elf", &Fahrenheit);
  ConvertToCelsius(); printf('\n\n $6.21f Fahrenheit de-
  grees are equivalent to %6.21f Celsius\n\n", Fahrenheit,
  Celsius);
}
void ConvertToCelsius(void)
{
Celsius=(Fahrenheit-32.0)*(5.0/9.0);
}
```


Lưu ý rằng "chương trình con" ConvertToCelsius đã được thực thi bằng cách sử dụng một hàm với void làm giá trị trả về ConvertToCelsius hoạt động trên các biến tổng thể Celsius và Fahrenheit (xem mục 5.3).

CHỦ ĐIỂM 5.3

SCOPE AND LIFETIME OF IDENTIFIERS
Phạm vi và thời gian tồn tại của các bộ nhận dạng

A useful concept in understanding the interaction of the different components of a program is that of the **scope** or **referencing environment**. This term defines the "visibility" of each variable, constant, function, or subprocedure. That is, the scope determines where a variable or constant can be used and where a function or subroutine can be referenced or invoked. The rules that allow a programmer to determine the referencing environment at any one time are called the **scope rules**. These rules may vary from language to language; however, in all languages we can differentiate between a global and local scope. When the visibility of a variable, constant, procedure, or function is confined to a section of the program, module, or form we say that the scope is local. Otherwise, we say that the scope is global. For example, if during the execution of a program a variable can be accessed at any time from any function and subroutine, the variable is said to be a global variable. However, if the variable can only be accessed within the function or subprocedure that defines it, the variable is said to be local. The following sections illustrate how these rules apply to C, C++, Java, and Visual Basic.

When a program starts executing, all of its global variables are allocated storage in main memory. Likewise, when a function or subroutine starts executing, all the local variables are allocated storage in main memory. The memory storage allocated to the global variables remains for the entire duration of the program. However, the storage allocated to the local variables of a function or subroutine is released as soon as the function or subroutine finishes executing. The term **lifetime** of a variable or constant refers to the duration of its storage allocation during the execution of the program. In other words, the lifetime determines how long a variable retains its assigned values. Using this definition we can say that the lifetime of global variables is for the entire duration of the program. The lifetime of all local variables, in general, is limited to the execution of the function or subprocedure that defines them.

Before addressing the issues of local versus global variables, let's consider when to use one over the other. The basic rule to remember is that the scope of a variable should be kept as narrow as possible. The main reasons for this are:

(1) Better organization in the program's structure. It has been recognized that the indiscriminate use of global variables may have undesirable side effects. The fact that a variable can be accessed from anywhere in the program also opens the possibility of being able to

modify the content of the variable anywhere in the program. If a global variable is changed at several places and its final result is not what we anticipated, it may be difficult to determine where it was changed last. To avoid this problem we need strictly to limit the number of places where a global variable can be changed. Ideally, global variables should be modified in only one place and one place only. This restriction will force us sometimes to look at the program design several times; however, in the experience of the authors it is a worthwhile effort.

(2) Conservation of memory. Since global variables must exist for the entire duration of the program, their storage locations in main memory must be maintained. This obviously increases the amount of memory that the program uses at any one time.

(3) Prevention of access to global variables. Most languages allow users to define variables with the same name at different places of the program. If a local variable is defined inside a procedure or function and its name is shared by a global variable then the global variable is no longer visible inside the function or procedure.

5.3.1 Scope of Identifiers in GC++ and Java - *Phạm vi của các bộ nhận dạng trong C/C++ và Java*

In C, C++, and Java all variables that are declared outside the body of functions are global; any module can use them. On the other hand, variables declared within the body of a function are always local to that function.

EXAMPLE 5.7 What is the scope of the variables in each of the functions shown below? What is the lifetime of the global and local variables?

```
#include<stdio.h>
void     print_double_sum (int, int);
void     print-triple-sum (int,int);
int total-value, /* This is a global variable */
int main ()
{
int valuel, value2;
printf('input two integers:');
scanf(*8d%d", &valuel, &value2);
print_double_sum (valuel, value2);
print_triple_sum (valuel,value2);
return (0);
}
```

```
void     print_double_sum (int val1, int val2)
total_value=2*(val1+val2); printf('The double value of the
  sum of %d and %d is =%d\n', val1, val2, total-value);
}
void     print- triple_sum (int val3, int val4)
{
total value=3*(val3+val4); printf('The triple value of the
  sum of. %d and %d is =8d\n1, val3, val4, total_value);
}
```

The variable total_value is defined before main and any other function in the program. The visibility of this variable is global. That is, it can be referenced from any other function in the program. Observe that this variable can be accessed within the functions print double sum and print triple_sum.

Variables valuel and value2 are defined within the main program and are local to that function. That is, if we try to reference either of these variables outside main we will get an error. Forinstance, if we try to use these two variables in any of the print statements of the functions print double_sum or print triple_sum we will get a compilation error.

Variables val1 and val2 are both defined within the function print_double_sum. They are local variables to this function only. That is, if we try to reference either of these variables outside this function we will get a compilation error.

Likewise, val3 and val4 are both defined within the function print triple sum. They are local variables to this function only. That is, if we try to reference either of these variables outside this function we will get a compilation error. Figure 5-4 illustrates this visibility of the variable in this example.

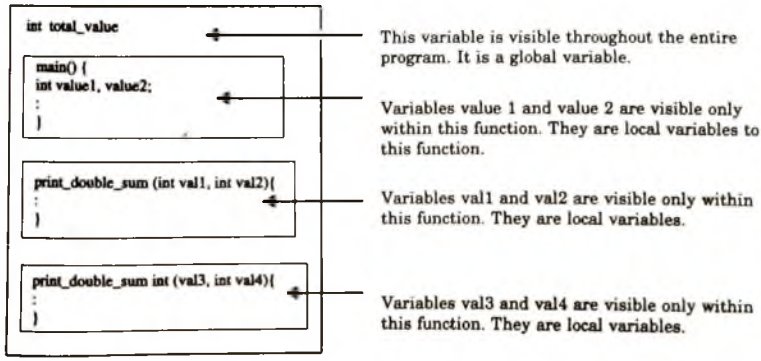


Fig. 5-4 Scope of variables for Example 5.4.

The global variable total _value has storage allocated to it for as long as the program is executing. Local variables val1, val2, val3, val4 are allocated storage as long as the functions in which they are defined are executing. Their storage is released when the functions finish executing. Variables value1 and value2 have storage allocated as long as the program is executing since they are declared in the main function.

Local variables declared with the **static** attribute retain their values even after the function or subroutine in which they are declared has finished executing. Static variables are only visible when the function or subprocedure in which they are defined is executing. These types of variables are generally used to keep track of running totals or in situations where it is necessary to make a decision based upon the number of times that a function has been called. Table 5-1 shows a partial classification of the scope and visibility of variables in the C/C++ languages.

Table 5-1 The Scope and Lifetime of C Variables.

Place of Declaration within the Program	Keyword	Visibility	Lifetime
Before all functions including main		Throughout the program	For the entire duration of the program
Inside a function		Only within the function	While function is executing
Inside a function	Static	Only within the function	For the entire duration of the program

EXAMPLE 5.8 The C function shown below illustrates the use of static variables. This function doubles its previous output every time that it gets called. However, the first time it gets called it prints its initial input parameters and returnd a zero.

```
#include<stdio.h>
int double_it(void);
int a=5, b=2;
void main()
{ printf("%d \n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
} /* end of main */
int double_it (void)
{static int previous_value;
 static int number_of_previous_calls=0;
 if (number_of_previous_calls==0) {
   printf("The initial values are %d and %d", a, b);
   number_of_previous_calls++;
   return(0); }
```

```
else if (number_of_previous_calls==1) {
  previous_value=2*a*b;
  number_of_previous_calls++;
  return (previous_value); }
 else {
  number_of_previous_calls++;
  previous_value=2*previous_value;
  return(previous_value); }
} /* end of function double_it*/
```

Notice that in the function double_it the variable number_of_previous_calls is declared and initialized to zero. The first time the function double_it is invoked, number_of_previous_calls has the value of zero since it was initialized to this value. After the first call to the function the static variable is no longer initialized and retains whatever value it was assigned during the last call to the function.

5.3.2 Scope of Identifiers in Visual Basic

5.3.2.1 Global Scope

The widest scope that you can give to any variable or constant in VB is through the use of the keyword **Public**. The general format declaration for variables or constants is:

Public **Const** ConstantName [As data type] = Value

or

Public VariableName [As dat type]

The square brackets indicate that the specification of the data type is optional. However, it always better specify the data type of all constants and variables. This book will follow this convention.

After a public constant or variable has been declared, it can be referenced anywhere in the project. However, there are restrictions as to where in your program you can declare Public variables or constants. Depending upon where the variable or constant is declared, there may be some restrictions on how they are referenced.

Public variables can only be declared in the General Declaration section of a Class, Standard, or Form module.

Public constants can only be declared in the General Doclatation section of a Standard module.

EXAMPLE 5.9 Assume that you declare a VB Public integer variable iTotalCount. What is the scope of this variable?

The answer to this question depends on where you declare the variable. If the variable was declared in a Standard module then all modules within the application can access the variable just by using its name. For example,

```
iTotalCount = 15
```

If the variable was declared in a Class or Form module, then the variable is considered as properties of the Class or Form. For example, if the variable is declared within a form called frmInitialForm, then the variable can be referenced as follows:

```
frmInitialForm.iTotalCount = 15
```

Notice that this feature will allow you to declare variables with identical names in all forms of the project. To reference any of these variables outside the form we would need to precede them with their corresponding form's name.

5.3.2.1.1 Private Variables

Variables **declared Private within** the General Declaration of a Form, Class, or Standard Module cannot be referenced outside the Form, Class, or Standard module where they were defined.

5.3.2.2 Local Scope

All variables declared inside an event procedure, general procedure, or function are considered local to that procedure or function. All variables declared using the keyword Dim are local variables. Another way of declaring local variables is through the use of the keyword Static.

EXAMPLE 5.10 The following subroutine changes to uppercase the text that the user types in a textbox called txtCategory.

```
Private Sub txtCategory_KeyPress (KeyAscii As Integer)
Dim stCharacter As String
  stCharacter=Chr(KeyAscii)
  stCharacter=Ucase(stCharacter)
  keyAscii=Asc(stCharacter)
End Sub
```

The KeyPress event occurs whenever the user presses a key that has an ASCII code. This procedure receives as its input an integer parameter that represents the ASCII code of the character pressed by the user. The name of this input parameter is KeyAscii.

The initial action of this procedure is to convert to a character the integer input parameter KeyAscii. This is carried out using the built-in function Chr which returns the ASCII character whose code is given by KeyAscii. The local variable stCharacter holds this character. The content of the variable stCharacter is then changed to an uppercase character using the built-in function Ucase. Finally, the integer ASCII code representing the uppercase character is assigned to the variable KeyAscii.

The code used in this procedure can be used whenever it is necessary to convert input text to all uppercase as the user types into a textbox. The string variable stCharacter is local to this subprocedure regardless of how many times the same code is used in similar procedures associated with other textboxes. The variable stCharacter cannot be referenced outside this procedure.

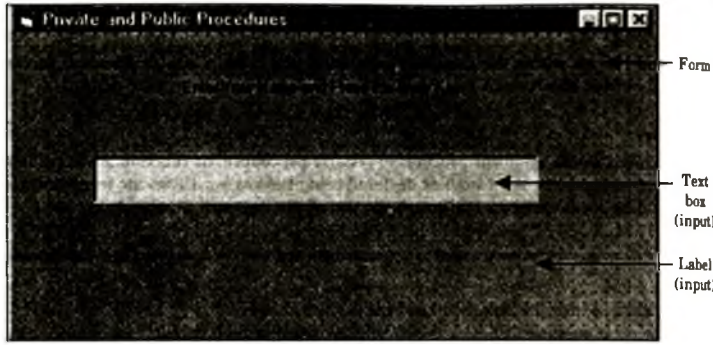
5.3.3 Scope of Procedures

In Visual Basic, procedures follow scope rules similar to the rules already defined for variables. There are two levels of scope rules for procedures within an application. Functions or procedures defined as Private can only be called from some other procedures or functions within the same Form, Class, or Standard module. Public procedures or functions can be called from anywhere within a project.

EXAMPLE 5.11 The Visual Basic procedure shown below accepts a name from a user and displays it all in uppercase as soon as the user presses the Enter key. This VB program uses public and private procedures. The pseudo algorithm for the program is shown below:

```
If KeyPress=ReturnKey Then
  Display Name In UpperCase
  Clear Name
Else
  Save Input Character
End I f
```

The implementation of this algorithm in Visual Basic uses the following form and objects.



- The textbox is named txtName.
- The label, where the name in uppercase is displayed, is named lblUpperName.
- The form is called frmName.

The general module section of the program should look like this:

```
Option Explicit
Public stName As String
Public Sub SaveInputCharacter (KeyAscii As Integer)
  If KeyAscii <> vbKeyBack Then
   stName=stName & UCase(Chr(KeyAscii))
  End If
End Sub
```

'This procedure clears name typed by the user in the textbox

```
Public Sub ClearName()
  stName=""
  frmName!txtName.Text=""
End Sub
```

In the form module the code section should look like this:

When the user presses the Enter Key (Return Key) the procedure displays the name of the user in caps on the label underneath the textbox. All other characters are converted into uppercase and saved until the user presses the return key.

```
Private Sub txtBoxl_KeyPress(KeyAscii As Integer)
If KeyAscii=vbKeyReturn Then
     DisplayNameInUpperCase
     ClearName
  Else
     SaveInputCharacter (KeyAscii)
  End If
End Sub
```

'This procedure, as its name indicates, displays the name typed by the user.

```
Private Sub DisplayNameInUpperCase()
  lblUpperName.Caption=stName
End Sub
```

Every time the user presses a key in the textbox the program checks to see if the pressed key is the Enter Key. Notice that we compare the code of the input key given (by keyAscii) with the global VB constant vbKeyRetum (ASCII code 13). If the Enter key is detected the program displays the message and clears the content of the textbox.

The Public procedure SaveInputCharacter changes every input character to its uppercase equivalent. The procedure uses the global variable stName to save the user's keystrokes. It does this by concatenating, that is, by putting together, each new input character with the previous input characters. In this case we have also ignored the backspace key. This way, if the user happens to use the backspace key, we avoid displaying a character that looks like this I as part of the user's name.

The Public procedure DisplayNameInUpper writes the content of the global variable stName into the caption of the label used to display the name.

The Public procedure ClearName sets the global variable stName and the content of the textbox to the Null string, therefore clearing their contents. Notice that to set the property of the textbox to the Null string, the textbox name has to be preceded by the form's name.

CHÚ THÍCH TỪ VỰNG

scope :	*phạm vi*
referencing environment :	*môi trường tham chiếu*
lifetime :	*thời gian sống*
static variables:	*các biến tĩnh*

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 5.3

5.3 PHẠM VI VÀ THỜI GIAN HOẠT ĐỘNG CỦA CÁC BỘ NHẬN DẠNG

Khái niệm hữu dụng để hiểu biết sự giao tiếp giữa các thành phần khác nhau của một chương trình đó là phạm vi hoặc môi trường tham chiếu. Mục này xác định tính "hiển thị" của mỗi biến, hằng, hàm hoặc thủ tục con. Có nghĩa rằng phạm vi sẽ xác định cho biết nơi mà một biến hoặc một hằng có thể được dùng và nơi mà một hàm hoặc trình con có thể được tham chiếu hoặc được viện dẫn. Các quy tắc cho phép nhà lập trình xác định môi trường tham chiếu tại bất cứ thời điểm nào gọi là các quy tắc định phạm vi. Những quy tắc này có thể thay đổi theo từng ngôn ngữ; tuy nhiên, trong hầu hết mọi ngôn ngữ chúng ta có thể phân biệt sự khác nhau giữa một phạm vi tổng thể và phạm vi cục bộ. Lúc tính hiển thị của một biến, một hằng, một thủ tục của một hàm được khẳng định cho một phần của chương trình, một modun, hoặc một dạng thì chúng ta bảo rằng đây là phạm vi cục bộ. Ngược lại, chúng ta bảo rằng phạm vi này là phạm vi tổng thể. Ví dụ, nếu trong suốt quá trình thực thi một chương trình, một biến có thể được truy cập vào bất cứ thời điểm nào từ bất cứ hàm nào và thủ tục con, thì biến nó được gọi là biến tổng thể. Tuy nhiên, nếu biến chỉ có thể được truy cập bên trong một hàm hoặc thủ tục con nhằm xác định nó, thì biến này được gọi là biến cục bộ. Phần sau đây minh họa cách mà những quy tắc này áp dụng cho C, C++, Java và Visual Basic.

Lúc một chương trình bắt đầu thực thi, tất cả các biến tổng thể của nó được cấp phát chỗ lưu trữ trong bộ nhớ chính. Cũng vậy trong một hàm hoặc một thường trình con bắt đầu thực thi thì tất cả các biến cục bộ đều được cấp phát chuẩn của nó trong bộ nhớ chính. Chỗ lưu trữ bộ nhớ được cấp phát cho các biến tổng thể sẽ lưu lại trong quá trình thực thi chương trình. Tuy nhiên, lưu trữ được cấp phát các biến cục bộ của một hàm hoặc thủ tục con thì bị chấm dứt ngay khi hàm hoặc thủ tục con hoàn tất việc thực thi. Thuật ngữ thời gian sống của một biến hoặc một hàm số hàm chỉ đến thời gian cấp phát lưu trữ của nó trong suốt quá trình thực thi chương trình. Nói cách khác, thời gian

sống xác định một biến được giữ lại bao lâu giá trị được gán cho nó. Bằng cách sử dụng định nghĩa này, chúng ta có thể bảo rằng thời gian sống của các biến tổng thể là toàn bộ thời gian hoạt động của chương trình. Còn thời gian sống của các biến cục bộ nói chung bị giới hạn trong thời gian thực thi hàm hoặc thủ tục con vốn xác định chúng.

Trước khi đề cập đến những biến cục bộ và biến tổng thể, thì chúng ta hãy xem xét lúc nào thì sử dụng loại này và lúc nào thì sử dụng loại khác. Quy tắc căn bản cần nhớ đó là phạm vi của một biến được giữa càng hẹp càng tốt. Các lý do chính của điều này là:

(1) Tổ chức tốt hơn cấu trúc của chương trình. Người ta nhận thấy rằng việc sử dụng ào ạt biến tổng thể có thể đưa ra những ảnh hưởng ngoài ý muốn. Sự thật là một biến có thể truy cập bất cứ nơi nào trong chương trình. Cũng mở ra một khả năng là chỉnh sửa một nội dung của biến đó bất cứ nơi nào trong chương trình. Nếu một biến tổng thể thì thay đổi ở một vài chỗ nào đó và kết quả sau cùng của nó không như bạn mong đợi, thì khó để xác định được nó bị thay đổi lần cuối ở đâu. Để tránh sự cố này, chúng ta phải nghiêm ngặt giới hạn số các chỗ nơi mà biến tổng thể có thể bị thay đổi. Xét điều kiện lý tưởng, các biến tổng thể nên được chỉnh sửa chỉ một nơi mà thôi. Hạn chế này sẽ tập trung chúng ta ở một nơi nào đó để xem xét việc thiết kế chương trình nhiều lần. Tuy nhiên, theo kinh nghiệm của tác giả thì đây là một nỗ lực có giá trị.

(2) Bảo trì được bộ nhớ. Bởi vì các biến tổng thể phải hiện diện trong suốt quá trình thực thi chương trình cho nên các vị trí lưu trữ chúng trong bộ nhớ chính phải được bảo quản. Rõ ràng rằng điều này sẽ làm gia tăng bộ nhớ mà chương trình đó sử dụng tại bất cứ thời điểm nào.

(3) Ngăn chặn sự truy cập vào các biến tổng thể. Hầu hết các ngôn ngữ đều cho phép người dùng sử dụng các biến với tên giống nhau với các vị trí khác nhau của chương trình. Nếu một biến cục bộ được xác định bên trong thủ tục hoặc hàm và tên của nó được chia sẻ bởi biến tổng thể thì biến tổng thể không còn được hiển thị bên trong hàm hoặc thủ tục nữa.

5.3.1 Phạm vi của các bộ định dạng trong C/C++ và Java

Trong C, C++ và Java tất cả các biến đều được khai báo bên ngoài nội dung của các hàm được gọi là biến tổng thể; bất cứ modun nào cũng có thể sử dụng chúng. Ngược lại, các biến được khai báo bên trong nội dung thì luôn luôn là cục bộ đối với hàm đó.

***VÍ DỤ 5.7** Phạm vi các biến trong mỗi một hàm được cho dưới đây là gì? Thời gian sống của biến tổng thể và biến cục bộ là gì?*

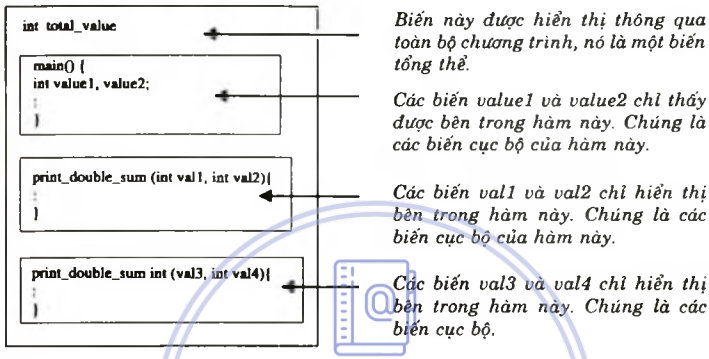
```
#include<stdio.h>
void     print_double_sum (int, int);
void     print-triple-sum (int,int);
int total-value, /* This is a global variable */
int main ()
{
int valuel, value2;
printf('input two integers:');
scanf(*8d%d", &valuel, &value2);
print_double_sum (valuel, value2);
print_triple_sum (valuel,value2);
return (0);
}
void     print_double_sum (int vall, int val2)
total_value=2*(vall+val2); printf('The double value of the
  sum of %d and %d is =%d\n', vall, val2, total-value);
}
void     print-triple_sum (int val3, int val4)
{
total value=3*(val3+val4); printf('The triple value of the
  sum of. %d and %d is =8d\nl, val3, val4, total_value);
}
```


Biến total_value được xác định trước hàm số chính và bất cứ hàm số nào khác trong chương trình. Tính hiển thị của biến này là tổng thể. Có nghĩa rằng nó có thể được tham chiếu trong bất cứ hàm nào khác trong chương trình. Quan sát ta thấy rằng biến này có thể được truy cập trong hàm print_double_sum và print_triple_sum.

Các biến value1 và value2 được xác định bên trong chương trình chính và là biến cục bộ đối với hàm đó. Điều này có nghĩa rằng nếu chúng ta tham chiếu mỗi trong số các biến này bên ngoài hàm chính thì ta sẽ nhận được lỗi. Nếu bạn sử dụng hai biến này trong bất cứ câu lệnh print nào của hàm print_double_sum hoặc print_triple_sum thì bạn sẽ nhận được lỗi biên soạn.

Các biến val1 và val2 cả hai biến đều được xác định bên trong hàm print_double_sum. Chúng là các biến cục bộ chỉ đối với hàm này thôi. Có nghĩa rằng, nếu bạn cố gắng tham chiếu mỗi một trong số các biến này bên ngoài hàm thì bạn sẽ nhận được một lỗi về biên soạn.

Cũng vậy val3 và val4 cả hai cũng được xác định bên trong hàm print_triple_sum. Chúng là các biến cục bộ chỉ cho hàm này. Có nghĩa rằng, nếu bạn thử tham chiếu mỗi trong số các biến này bên ngoài hàm thì sẽ nhận được một lỗi về biên soạn. Hình 5.4 minh họa sự hiển thị của biến này trong ví dụ.



***Hình 5.4** Các phạm vi của các biến trong ví dụ 5.4*

Biến tổng thể total_value được cấp phát chỗ lưu trữ cho nó ngay khi chương trình thực thi. Các biến cục bộ val1, val2, val3, val4 được cấp phát chỗ lưu trữ ngay khi các hàm mà chúng xác định được thực thi. Chỗ lưu trữ của chúng bị chấm dứt lúc các hàm hoàn thành việc thực thi. Các biến value1 và value2 có chỗ lưu trữ được cấp phát khi chương trình thực thi bởi vì chúng được khai báo trong hàm chính.

Các biến cục bộ được khai báo với thuộc tính static vẫn giữ lại giá trị của chúng thậm chí sau khi chương trình hoặc thủ tục con mà chúng được khai báo đã thực thi xong. Các biến static chỉ thấy được lúc hàm hoặc thủ tục con mà chúng định nghĩa được thực thi. Những kiểu biến này thường theo dõi toàn bộ quy trình hoạt động hoặc những tình huống nơi chúng ta cần đưa ra quyết định dựa trên số lần mà một hàm được gọi. Bảng 5.1 minh họa lớp từng phần của phạm vi và hiển thị của các biến tong các ngôn ngữ C/C++.

Bảng 5.1 *Phạm vi và thời gian sống của các biến C*

Vị trí khai báo bên trong chương trình	*Từ khóa*	*Tính thấy được*	*Thời gian sống*
Trước tất cả các hàm kể cả hàm chính		*Thông qua chương trình*	*Dành cho suốt thời gian thực thi chương trình*
Bên trong một hàm		*Chỉ bên trong hàm đó*	*Trong khi hàm đang thực thi*
Bên trong một hàm	*Static*	*Chỉ bên trong hàm đó*	*Trong suốt thời gian thực thi chương trình*

VÍ DỤ 5.8 *Hàm C minh họa dưới đây sẽ trình bày cách sử dụng các biến tĩnh. hàm này tăng gấp đôi kết quả xuất trước đó mỗi khi nó được gọi. Tuy nhiên, lần đầu tiên khi nó nhận được cuộc gọi thì nó in các tham số khởi nhập và trả về một zero.*

```
#include<stdio.h>
int double_it(void);
int a=5, b=2;
void main()
{ printf("%d \n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
  printf("%d\n", double_it());
} /* end of main */
int double_it (void)
{static int previous_value;
 static int number_of_previous_calls=0;
 if (number_of_previous_calls==0) {
   printf("The initial values are %d and %d", a, b);
   number_of_previous_calls++;
   return(0); }
else if (number_of_previous_calls==1) {
   previous_value=2*a*b;
   number_of_previous_calls++;
   return (previous_value); }
 else {
   number_of_previous_calls++;
   previous_value=2*previous_value;
   return(previous_value); }
} /* end of function double_it*/
```

Lưu ý rằng trong hàm double_it biến number_of_previous_calls được khai báo và được khởi tạo sau zero. Lần đầu tiên hàm double_it được viện dẫn, number_of_previous_calls có giá trị bằng zero bởi vì nó được khởi tạo sang giá trị này. Sau lần gọi đầu tiên cho hàm này, biến tĩnh không còn được khởi tạo và được giữ lại bất kỳ giá trị nào mà nó được gán trong suốt cuộc gọi sau cùng đến hàm.

5.3.2 Phạm vi của các bộ định dạng trong Visual Basic

5.3.2.1 Phạm vi tổng thể

Phạm vi rộng nhất mà bạn có thể cung cấp cho bất kỳ biến hoặc hằng nào cho VB thông qua việc sử dụng từ khóa public. Khai báo dạng tổng quát dành cho các biến và các hằng là:

```
Public Const ConstantName [As data type] = Value
```

hoặc

```
Public VariableName [As dat type]
```

Dấu móc vuông chỉ ra rằng đặc trưng của kiểu dữ liệu là tùy ý. Tuy nhiên, tốt hơn ta nên chỉ định kiểu dữ liệu của tất cả các hàm và các biến. Sách này sẽ tuân theo quy ước này.

Sau khi một hằng công cộng hoặc một biến công cộng được khai báo, thì nó có thể được tham chiếu ở bất cứ nơi nào trong đồ án. Tuy nhiên, cũng có một số hạn chế ở nơi chương trình của bạn có thể được khai báo các biến Public hoặc các hằng. Chỉ phụ thuộc vào nơi mà hằng hoặc biến được khai báo có thể có một vài giới hạn về cách mà chúng được tham chiếu.

Các biến công cộng chỉ có thể được khai báo trong mục khai báo tổng quát của Class, Standard, hoặc Form module.

Các hằng công cộng chỉ có thể được khai báo trong mục tổng quát của Standard module.

***VÍ DỤ 5.9** Giả sử rằng bạn đang khai báo biến nguyên VB Public iTotalCount. Phạm vi của biến này là gì?*

Câu trả lời cho câu hỏi này phụ thuộc vào khi mà bạn khai báo biến. Nếu biến khai báo trong một Standard module thì tất cả các module bên trong trình ứng dụng có thể truy cập vào biến bằng cách sử dụng tên của nó. Ví dụ,

```
iTotalCount = 15
```

Nếu biến được khai báo trong một Class hoặc Form module, thì biến được xem như là tính chất của Class hoặc Form. Ví dụ, nếu biến này được khai báo bên trong một form có tên là frmInitialForm, thì biến này có thể được tham chiếu dưới tên:

```
frmInitialForm.iTotalCount = 15
```

Lưu ý rằng đặc tính này sẽ cho phép bạn khai báo các biến với các tên đồng nhất trong tất cả các dạng đề án. Để tham chiếu bất cứ biến nào bên ngoài form chúng ta cần phải đặt trước chúng tên form tương ứng.

***5.3.2.1.1** Các biến riêng tư*

Các biến được khai báo Private bên trong khai báo tổng quát của một Form, Class hoặc Standard Module không thể được tham chiếu bên ngoài Form, Class, hoặc Standard module nơi mà chúng được xác định.

***5.3.2.2** Phạm vi cục bộ*

Tất cả các biến được khai báo bên trong một thủ tục biến cố, thủ tục tổng quát, hoặc một hàm được xem như là biến cục bộ đối với thủ tục hoặc hàm đó. Tất cả các biến được khai báo bằng cách sử dụng từ khóa Dim đều là các biến cục bộ. Một cách khác để khai báo các biến cục bộ đó là thông qua việc sử dụng từ khóa Static.

***VÍ DỤ 5.10** Trình con sau đây thay đổi kiểu chữ hoa của text mà người dùng nhập vào trong textbox gọi là được gọi là txtCategory.*

```
Private Sub txtCategory_KeyPress (KeyAscii As Integer)
Dim stCharacter As String
  stCharacter=Chr(KeyAscii)
  stCharacter=Ucase(stCharacter)
  keyAscii=Asc(stCharacter)
End Sub
```

Biến cố KeyPress xảy ra lúc người dùng nhấn bàn phím có mã ASCII. Thủ tục này nhận nó dưới dạng dữ liệu nhập một tham số nguyên biểu thị mã ASCII của ký tự mà người dùng nhấn lên. Tên của tham số nhập này là KeyAscii.

Hành động khởi tạo của thủ tục này được chuyển sang một ký tự tham số nhập số nguyên KeyAscii. Nó được thực thi bằng cách sử dụng hàm tạo sẵn Chr trả về ký tựASCII mã của nó được cho bởi KeyAscii. Biến cục bộ stCharacter giữ ký tự này. Nội dung của biến stCharacter sau đó được thay đổi sang ký tự kiểu chữ hoa bằng cách sử dụng hàm tạo sẵn Ucase. Sau cùng, số nguyên mã ASCII trình bày ký tự kiểu in hoa được gán cho biến KeyAscii.

Mã này được dùng trong thủ tục có thể được dùng trong bất cứ nơi nào cần thiết để biến đổi text nhập sang tất cả các chữ in hoa khi người dùng gõ nhập vào textbox. Biến chuỗi stCharacter là cục bộ đối với thủ tục con này bất kể bao nhiêu lần số mã giống nhau được dùng trong thủ tục giống nhau liên quan đến text box khác. Biến stCharacter

không thể được tham chiếu bên ngoài thủ tục này.

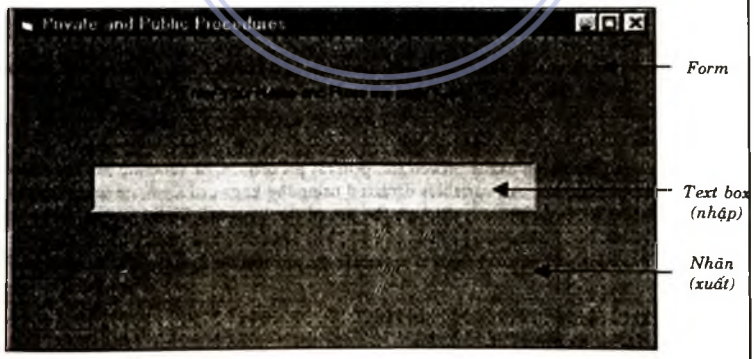
5.3.3 Phạm vi của các thủ tục

Trong Visual Basic, các thủ tục tuân theo các quy tắc phạm vi tương tự như các quy tắc đã được xác định cho các biến. Có hai mức quy tắc phạm vi dành cho thủ tục bên trong một trình ứng dụng. Các hàm hoặc các thủ tục được xác định dưới dạng Private chỉ có thể được gọi từ một vài thủ tục hoặc hàm khác bên trong cùng Form Class hoặc Standard module. Các thủ tục hoặc các hàm Public có thể được gọi bất cứ nơi nào bên trong đề án.

VÍ DỤ 5.11 *Thủ tục Visual Basic minh họa dưới đây nhận tên của một người dùng và hiển thị nó theo kiểu chữ in hoa ngay khi người dùng nhấn phím Enter. Chương trình VB sử dụng các thủ tục public và private. Thuật toán mã giả của chương trình được minh hoạ dưới đây.*

```
If KeyPress=ReturnKey Then
  Display Name In UpperCase
  Clear Name
Else
  Save Input Character
End If
```


Việc thực thi thuật toán này trong Visual Basic sử dụng form và đối tượng sau đây.



- *Textbox được đặt tên txtName.*
- *Nhãn, là nơi theo kiểu chữ in hoa được hiển thị, được đặt tên là iblUpperName.*

- *Form được đặt tên là frmName.*

Module tổng quát của chương trình giống như dưới đây:

```
Option Explicit
Public stName As String
Public Sub SaveInputCharacter (KeyAscii As Integer)
  If KeyAscii <> vbKeyBack Then
   stName=stName & UCase(Chr(KeyAscii))
  End If
End Sub
```

Thủ tục này xóa tên được người dùng gõ nhập trong textbox.

```
Public Sub ClearName()
  stName=""
  frmName!txtName.Text=""
End Sub
```

Trong module form, phần mã sẽ giống như dưới đây:

Lúc người dùng nhấn phím Enter (phím Return) thì thủ tục hiển thị tên của người dùng theo chữ in hoa trên nhãn ngay bên dưới textbox. Tất cả các ký tự khác đều được biến đổi sang ký tự in hoa và được lưu cho đến khi người dùng nhấn phím return.

```
Private Sub txtBoxl_KeyPress(KeyAscii As Integer)
If KeyAscii=vbKeyReturn Then
      DisplayNameInUpperCase
      ClearName
  Else
      SaveInputCharacter (KeyAscii)
  End If
End Sub
```

Thủ tục này, như tên đã ngụ ý, sẽ hiển thị tên mà người dùng gõ nhập.

```
Private Sub DisplayNameInUpperCase()
  lblUpperName.Caption=stName
End Sub
```

Mỗi lần người dùng nhấn một phím trong textbox thì chương trình kiểm tra xem thử phím được nhấn có phải là phím Enter hay không.Lưu ý rằng chúng ta so sánh mã của phím nhập vào được cho bởi keyAscii)

với hằng số VB tổng thể vbKeyReturn (ASCII mã 13). Nếu phím Enter được dò tìm thì chương trình niển thị thông điệp và xóa nội dung của textbox.

Thủ tục Public SaveInputCharacter thay đổi mỗi một ký tự nhập sang kiểu chữ in hoa tương đương. Thủ tục này sử dụng biến tổng thể stName để lưu các phím của người dùng. Nó thực hiện điều này bằng cách đặt lại với nhau mỗi ký tự nhập mới với các ký tự nhập trước đó. Trong trường hợp này chúng ta cũng bỏ qua phím backspace. Theo cách này, nếu người dùng tình cờ sử dụng phím backspace thì chúng ta tránh được việc hiển thị một ký tự giống như | là một phần của tên người dùng.

Thủ tục Public DisplayNameInUpper sẽ viết nội dung biến tổng thể stName vào phần chú giải của nhãn dùng để hiển thị tên.

Thủ tục Public ClearName xác lập biến tổng thể stName và nội dung của textbox sang chuỗi Null, do đó xóa nội dung của nó. Lưu ý rằng để xác định tính chất của textbox sang chuỗi Null, tên textbox phải có tên của form đứng trước.

CHỦ ĐIỂM 5.4

PARAMETER-PASSING MECHANISMS
Cơ chế truyền tham số

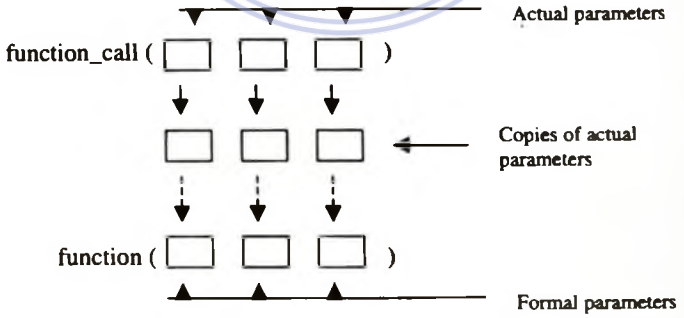
The term **parameter-passing** mechanism refers to the different ways in which parameters are passed to or from a function or procedure. As we indicated in Section 5.1, a procedure or function can call any other procedure that is visible to it. The procedure that makes the call is called the "calling procedure" and the procedure that is invoked is the "called procedure." There are two basic models of how data transfers take place between functions or procedures. In one model, actual values are physically moved from the calling function or procedure to the called function or procedure. In the second model, an access path is transmitted to the called function or procedure. What get transmitted are the addresses in memory where the actual parameters are.

5.4.1 The Pass-by-Value Mechanism - *Cơ chế truyền theo giá trị*

When an actual parameter is **passed by value,** the value of the actual parameter is used to initialize the corresponding actual parameter. In other words, the formal parameter receives a copy of the value of the actual parameter. Any changes made to the value of this formal parameter do not affect the value of the actual parameter.

Pass-by-value is the parameter-passing mechanism by default in C, C++, and Java.

The following diagram illustrates this parameter-passing mechanism:



The called function only receives a copy of the actual parameters. Therefore, any changes made to the formal parameters do not affect the original values of the actual parameters.

EXAMPLE 5.12 The following C program reads in the numerical value for n and doubles it.

```
#include <stdio.h>
void DoubleNumber(int);
void main()
{
int iValue; printf("\nEnter an integer value: ");
scanf("$d", &iValue);
printf("\n\nThe actual parameter is %d", ivalue);
DoubleNumber(iValue);
printf("\n\nThe value of the actual parameter after the func-
  tion call is %d\n\n", iValue);
} /* End of main function*/
void DoubleNumber(int n)
{
  printf("\n\nThe value assigned to the formal parameter is
  %d\n\n", n);
  n=2*n;
  printf("\n\nThe double of the input parameter is %d\n\n",
  n);
} /*End of the function DoubleNumber */
```


The main function calls the procedure DoubleNumber to calculate the double of the input number Walue. The value of iValue is printed before and after the function is called. As the user can verify, the value of iValue is unchanged. Notice that the formal parameter n is changed in the called function.

EXAMPLE 5.13 Write Visual Basic code that calculates the absolute value of a given number n. The main form is shown below.

- The form is called frmAbsolute.
- The textboxes are called txtInput and txtOutput respectively.

The code for implementing this program is:

```
Private Sub txtInput_KeyPress(KeyAscii As Integer)
  Dim stInput As String
  If KeyAscii=vbKeyReturn Then
     If IsNumeric(txtInput.Text) Then
       txtInput.BackColor=vbWhite
       CalculateAbsoluteValue (Val(txtInput.Text))
     Else
       MsgBox "Input is not numeric", vbInformation, "In-
valid data"
     txtInput.Text='-txtInput.BackColor=vbYellow
     End If
  End If
End Sub
Private Sub Cal culateAbsoluteValue(ByVal InputValue As
  Single)
If InputValue>=0 Then
txtOutput.Text=Str(InputValue)
Else
InputValue=-InputValue
txtOutput.Text=Str(InputValue) End If
End Sub
```


The subprocedure txtInputKeyPress determines whether the input value is numeric. If the input value is numeric the subprocedure CalculateAbsoluteValue determines the absolute value of the given input. If the input value is not numeric the program displays an error message and set the background of the textbox to yellow. When the user enters a correct value the background color is set to white. The built-function IsNumeric, as its name indicates, is used to verify that the user's input is numeric. The user is notified of any invalid input by means of a message box.

Notice that the formal parameter InputValue is declared with the keyword ByVal. This way the subprocedure CalculateAbsoluteValue only gets a copy of the input value. The actual parameter is not changed, even though the formal parameter is assigned a value with an opposite sign inside the

procedure. The formal parameter InputValue has been declared as a Single data type. This data type is generally used for storing values that may contain a fractional part.

5.4.2 The Pass-By-Reference Mechanism - *Cơ chế truyền theo tham chiếu*

When an actual parameter is passed to a function or subprocedure using the **pass-by-reference** mechanism, the address of the actual parameter is passed to the formal parameter. This way, the actual and formal parameters refer to the same memory location. As a consequence, any changes made to the formal parameter affect the value of the actual parameter. This is the default mechanism for passing arguments in VB. In C and C++ this mechanism is implemented using "pointers." Java does not use these pointers. A pointer is nothing more than a variable that can hold the address of an object, which can be either a variable, subprocedure, or function.

As we indicated in Section 1.2.1, every location has a unique address associated with it. The address of a variable remains constant throughout the execution of a program, whereas the content of the variable may change many times while the program is executing.

EXAMPLE 5.14 The following C program illustrates the difference between the address and content of a variable.

```
#include<stdio.h>
void main ()
{ int value1;
  printf("\nThe address of the integer variable value1 is:");
  printf("%x",&value1);
  value1=15;
  printf("\nThe new value of variable value1 is:");
  printf("%d",value1);
  value1=15;
  printf("\nThe new value of variable value1 is:");
  printf("%d",value1);
  printf("\nThe address of the integer variable value1 is: ");
  printf("%x\n\n",&value1); }
```

The output of this program is:

The address of the integer variable value1 is: 65fdf4

The new value of variable is: 15

The address of the integer variable value1 is: 65fdf4

Notice that the address of the variable is printed using the address operator &. This operator, when used in conjunction of a variable as illustrated in this program, is a reference to the address of the memory location occupied by the variable. The conversion specification %x used in the control string of the first and third call to printf indicates that the value should be printed in hexadecimal. The content of the variable changes twice during the execution of the program but its address remains the same. The & operator is commonly used in the scanf function.

5.4.3 Pointers - *Các con trỏ*

In C and C++ each data type has a corresponding pointer data type. For example, the statement shown below declares a pointer to a variable of type integer. This variable is capable of holding the address of any integer variable used throughout the program.

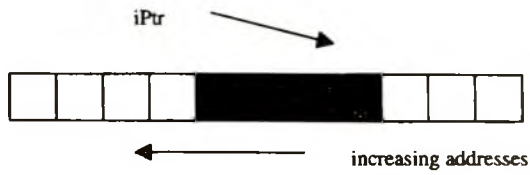
```
int * iPtr; /* declaring an integer to an integer variable */
```

The * placed to the immediate left of the variable iPtr is an unary operator called the **indirection or dereferencing** operator. If we assume that an integer variable called iValue has been defined in the program, the statement shown below assigns the address of this variable to the pointer variable. Notice the use of the address operator to reference the address of the variable iValue.

```
iPtr = &iValue; /* assigning the address of variable iValue.to iPtr */
```

In this case we say that the variable iPtr "points to" the memory location iValue. The following diagram illustrates this.

The pointer variable points to the address of the first byte of the memory location of the integer variable iValue (shown in the diagram as occupying four consecutive bytes).



When the dereferencing operator is used with a pointer variable, it accesses the content of the variable that it points to. The following section of code illustrates the use of both operators.

```
int x, y;          /* declaration of variables x and y */
int * iPtr;        /* pointer to an integer variable */
```

```
x = 10;          /* new content of variable x */
iPtr = &x        /* variable iPtr points to variable x */
y = *iPtr;       /* variable y is assigned the content of variable x */
```

In this last statement *iPtr refers to the content of variable x. The value of this variable, 10, is assigned to variable y.

5.4.4 Function and Procedure Arguments Using Pointers - *Các đối số hàm và thủ tục sử dụng các con trỏ*

In languages like C++ and C the pass-by-reference mechanism is implemented using pointers. In this method the address of the actual parameter is passed to a subprogram. The address itself is not modified, but the called function or procedure may modify the content of that address. Whenever an address is passed as an actual parameter, the corresponding formal parameter in the subprogram must be declared as a pointer.

EXAMPLE 5.15 The following C program swaps the value of two given integer variables using pointers.

```
#include <stdio.h>
void swap_values(int *, int *); /* function prototype */
int a=5, b=2;
void main()
{
printf("\nThe initial values of the variables a and b are %d
  and %d respectively", a,b);
swap-values (&a,&b); /* swap content of variables */
printf('\n\nThe final values of the variables a and b are %d
  and %d respectively\n', a,b); } /* end of main */
void swap_values (int * iPtr1, int * iPtr2)
{
int temp;
temp=*iPtr1;
*iPtr1=*iPtr2;'
*iPtr2=temp;}
```

Notice that the addresses of the actual parameters were passed to the subprocedure using the dereferencing operator and that the corresponding formal parameters were defined as pointers. In the function prototype **void swap_values (int *, int *)**; the data type of both parameters is defined as "pointers to integers." There is no mention of the name of the parameters.

This prototype could have been defined as shown below without affecting the execution of the program.

```
void swap_values(int * x, int * y);
```

As we indicated before, the role of the variables x and y in the prototype is that of placeholders. No formal or actual parameter has to be named after any of these variables.

EXAMPLE 5.16 The following program illustrates the swap function from Example 5.15 above in C++. It shows an alternate way to pass arguments by reference. It uses the &, or address operator, instead of the * pointer.

```
void swap_values (int& iPtr1, int& iPtr2)
{
int temp;
temp=iPtr1;
iPtr1=iPtr2;
iPtr2=temp;}
```

Notice that C++ formal parameters use the & to indicate that the memory location is being passed from the calling program. The variables sent as actual parameters will be changed. The line needed to call this function does not send the pointers, but the variables themselves.

```
swap values (a,b); /* swap content of variables */
```

In Visual Basic the default parameter mechanism is pass-by-reference. The following example illustrates this.

EXAMPLE 5.17 The following procedure changes ASCII lowercase characters to uppercase.

Assume that there is a textbox called txtUserInput. The subprocedure associated with the event KeyPress is shown below. As its name indicates, this event occurs only when a user presses a key that has a corresponding ASCII code.

```
Private Sub txtuserInput_KeyPress (KeyAscii As Integer)
Dim Character as String
Character=Chr(KeyAscii)          'convert input value to character
Character=Ucase(Character) 'convert character to uppercase
KeyAscii=Asc(Character) 'convert character to ASCII code
End Sub
```

The input value to this procedure is the integer variable KeyAscii. This variable represents the ASCII code of the key pressed by the user.

The subprocedure initially converts the input integer value to a character using the built-in function Chr(). This character is then transformed to an uppercase character using the built-in function Ucase(). Finally, the uppercase character is converted to its equivalent ASCII code using the built-in function Asc(). Although we have used three separate statements to perform these transformations, it is possible to combine all three functions in a single statement as follows

```
KeyAscii = Asc(Ucase(Chr(KeyAscii)))
```

Notice that the changes to the input parameter are reflected immediately, since the actual and formal parameters are referring to the same address.

5.4.5 Parameter Passing in Java - *Việc truyền tham số trong Java*

Java is not a procedural language like C or C++. In Java, every function, or method, is part of a larger class. See Chapter 8 for a full explanation of the Java language class structure. Java functions look and behave in a similar way as C++ functions, except that all class objects are passed by reference, and all simple types are passed by value. Neither the ampersand (&) nor the asterisk (*) is ever used. Only simple Java functions will be illustrated here, not the class calling code.

EXAMPLE 5.18 Write a Java function to return a double value representing the area of a circle, given the integer radius as a parameter.

Using 3.14 as the value of π and the area of a circle as π radius2. The function may look like this.

```
public double FindAreaCircle(int radius)
{
  final double PI=3.14;
  return (PI*radius*radius);
}
```

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 5.4

5.4 CƠ CHẾ TRUYỀN THAM SỐ

Thuật ngữ cơ chế truyền tham số đề cập đến các cách khác nhau mà qua đó các tham số được truyền đi hoặc đến từ một hàm hoặc một thủ tục. Như đã đề cập trong mục 5.1, một thủ tục hoặc một hàm có thể có thể gọi bất kỳ thủ tục nào khác mà nó nhìn thấy. Thủ tục thực hiện cuộc gọi được gọi là "thủ tục gọi" và thủ tục được viện dẫn là "thủ tục

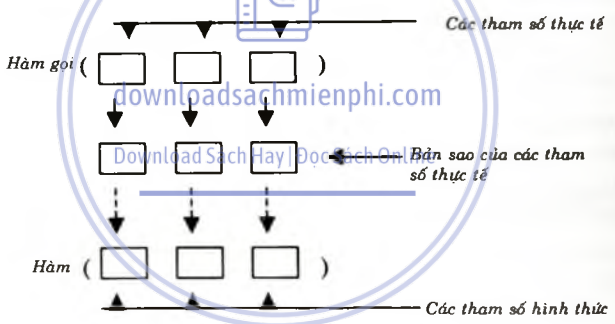
gọi". Có hai mẫu căn bản về cách dữ liệu truyền giữa các hàm và các thủ tục. Ở trong mẫu thứ nhất, các giá trị thực tế thì hàm gọi và thủ tục gọi cho đến hàm và thủ tục được gọi. Trong mẫu thứ hai một đường dẫn truy cập được truyền sang cho hàm hoặc thủ tục gọi. Những gì được truyền chính là các địa chỉ trong bộ nhớ nơi mà các tham số thật sự đang có.

5.4.1. Cơ cấu truyền theo giá trị

Lúc một tham số thực sự được truyền theo giá trị của tham số thực tế để khởi tạo tham số thực tế tương ứng. Nói cách khác tham số hình thức nhận một bản sao giá trị trong tham số thực tế. Bất cứ thay đổi nào thực hiện trên giá trị của tham số hình thức này đều không ảnh hưởng đến giá trị của tham số thực tế.

Sự truyền theo giá trị chính là một cơ cấu truyền tham số theo mặc định trong C, C++, và Java.

Sơ đồ sau đây minh họa cơ cấu truyền tham số:



Hàm được gọi chỉ nhận một bản sao của tham số thực tế. Do đó bất cứ thay đổi nào thực hiện cho các tham số hình thức đều không ảnh hưởng đến giá trị gốc của các tham số thực tế.

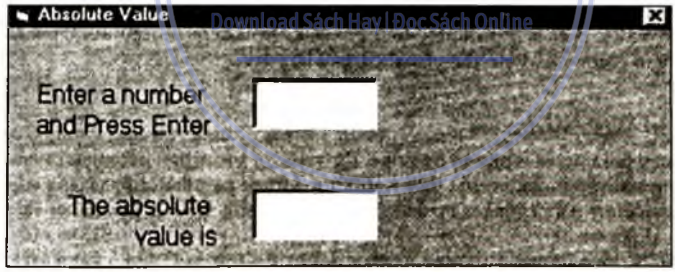
***VÍ DỤ 5.12** Chương trình C sau đây đọc giá trị số đành cho n rồi nhân đôi nó.*

```
#include <stdio.h>
void DoubleNumber(int);
void main()
{
int iValue; printf("\nEnter an integer value: ");
scanf("%d", &iValue);
```

```
printf("\n\nThe actual parameter is %d", ivalue);
DoubleNumber(iValue);
printf("\n\nThe value of the actual parameter after the func-
  tion call is %d\n\n", iValue);
} /* End of main function*/
void DoubleNumber(int n)
{
  printf("\n\nThe value assigned to the formal parameter is
  %d\n\n", n);
  n=2*n;
  printf("\n\nThe double of the input parameter is %d\n\n",
  n);
} /*End of the function DoubleNumber */
```

Hàm số chính gọi thủ tục DoubleNumber để tính toán phần gấp đôi số đầu vào iValue. Giá trị của iValue được in trước và sau khi hàm được gọi nhưng người dùng có thể kiểm nghiệm, giá trị iValue không bị thay đổi. Lưu ý rằng tham số hình thức n bị thay đổi trong hàm gọi.

VÍ DỤ 5.13. *Hãy viết mã Visual Basic để tính giá trị tuyệt đối của một n đã cho. Dạng chính được minh họa dưới đây.*



- *Dạng này có tên là frmAbsolute.*
- *Textbox có tên là txtInput và txtOutput tương ứng.*

Mã để thực thi chương trình này là:

```
Private Sub txtInput_KeyPress(KeyAscii As Integer)
  Dim stInput As String
  If KeyAscii=vbKeyReturn Then
     If IsNumeric(txtInput.Text) Then
        txtInput.BackColor=vbWhite
```

```
        CalculateAbsoluteValue (Val(txtInput.Text))
      Else
        MsgBox "Input is not numeric", vbInformation, "In-
  valid data"
      txtInput.Text=" "
      txtInput.BackColor=vbYellow
      End If
    End If
  End Sub
  Private Sub Cal culateAbsoluteValue(ByVal InputValue As
    Single)
  If InputValue>=0 Then
  txtOutput.Text=Str(InputValue)
  Else
  InputValue=-InputValue
  txtOutput.Text=Str(InputValue) End If
  End Sub
```


Thủ tục con txtInputKeyPress xác định cho biết giá trị đầu vào có phải là số hay không. Nếu giá trị đầu vào là một con số thì thủ tục con CalculateAbsoluteValue sẽ xác định giá trị tuyệt đối của đầu vào đã cho. Nếu giá trị đầu vào không phải là số thì chương trình hiển thị một thông báo lỗi và xác lập lại nền của textbox sang màu vàng. Lúc người dùng nhập vào một giá trị đúng thì màu nền được xác lập sang màu trắng. Hàm được tạo sẵn IsNumeric, như tên nó đã gợi ý, được dùng kiểm nghiệm rằng giá trị đầu vào của người dùng là số hay không. Người dùng được nhắc nhở về bất cứ giá trị nào không đúng bằng một hộp thông điệp.

Lưu ý rằng tham số hình thức InputValue được khai báo với từ khóa Byval. Theo cách này thì thủ tục con CalculateAbsoluteValue chỉ nhận được một bản sao của giá trị đầu vào, tham số thực tế không bị thay đổi thậm chí tham số thực tế đầu vào gán với một giá trị với một dấu bên trong số nghịch. Tham số hình thức được gán vào một giá trị với một dấu bên trong thủ tục. Tham số hình thức InputValue được thay thế bằng một kiểu dữ liệu Single. Kiểu dữ liệu này thường được dùng để lưu trữ các giá trị có thể chứa một phần thập phân.

5.4.2. Cơ cấu truyền theo tham chiếu

Lúc một tham số thực tế truyền cho một hàm hoặc một thủ tục con bằng cách sử dụng cơ cấu truyền theo tham chiếu, thì địa chỉ của tham số thực sự được truyền đến thum chiếu hình thức. Theo cách

này tham số thực tế và các tham chiếu hình thức đều tham chiếu đến vị trí bộ nhớ giống nhau. Ở bất cứ tham số nào được thực hiện cho tham số hình thức đều ảnh hưởng đến tham số thực tế trước đây, cơ cấu mặc định dành cho việc truyền các đối số trong VB. Trong C và C++ cơ cấu này được thực thi bằng cách sử dụng "các pointer"Java không sử dụng những pointer này. Một pointer không gì khác hơn là một biến có thể giữ địa chỉ của một đối tượng, nó có thể thủ tục con hoặc một hàm.

Như chúng ta đã chỉ định trong mục 1.2.1 mỗi một vị trí đều có một địa chỉ duy nhất liên kết với nó. Địa chỉ của một biến vẫn giữ không đổi, mặc dù trong suốt quy trình thực thi một chương trình trong khi nội dung của biến có thể thay đổi nhiều lần trong khi một chương trình đang được thực thi.

***VÍ DỤ 5.14.** Chương trình C sau đây minh họa sự khác biệt giữa địa chỉ và nội dung của một biến.*

```
#include<stdio.h>
void main ()
{ int value1;
  printf("\nThe address of the integer variable value1 is:");
  printf("%x",&value1);
  value1=15;
  printf("\nThe new value of variable value1 is:");
  printf("%d",value1);
  value1=15;
  printf("\nThe new value of variable value1 is:");
  printf("%d",value1);
  printf("\nThe address of the integer variable value1 is: ");
  printf("%x\n\n",&value1); }
```


Đầu ra của chương trình này là:

Địa chỉ của biến nguyên value1 là: 65fdf4

Giá trị mới của biến là: 15

Địa chỉ của biến nguyên Value1 là: 65fdf4

Lưu ý rằng địa chỉ của biến được in bằng cách sử dụng toán tử địa chỉ dấu &. Toán tử này lúc được dùng liên kết với một biến như minh họa trong chương trình này, thì là một tham chiếu đến địa chỉ của vị trí bộ nhớ bị chiếm giữ bởi biến. Đặc trưng biến đổi %x được dùng trong chuỗi điều khiển của cuộc gọi thứ nhất và thứ ba ngoài ra printf được chỉ định rằng giá trị nên nên được in theo hệ thống lục phân. Nội dung của biến thay đổi hai lần trong khi thực thi chương trình, nhưng địa chỉ của nó thì vẫn giữ không đổi. Toán tử thường được dùng trong hàm scanf.

Các pointer (con trỏ)

Trong C++ mỗi kiểu dữ liệu đều có một kiểu dữ liệu pointe tương ứng. Ví dụ câu lệnh được minh họa dưới đây khai báo một pointe biến thuộc kiểu nguyên. Biến này có khả năng giữ địa chỉ của bất kỳ giá trị nguyên nào được dùng trong suốt chương trình.

```
int*iPtr;/* các khai báo một số nguyên sang một biến nguyên */
```

*dấu * được đặt ngay bên trái của iPtr là một toán tử được gọi là indirection hoặc toán tử dereferencing. Nếu chúng ta giả sử rằng một biến nguyên có tên là iValue được xác định trong chương trình, thì câu lệnh được minh họa dưới đây sẽ gán địa chỉ này của biến này cho biến pointer. Lưu ý việc sử dụng toán tử địa chỉ để tham chiếu đến địa chỉ của biến iValue.*

```
iPtr = &iValue;  /* sẽ gán địa chỉ của biến iValue đến iPtr* /.
```

Trong trường hợp này chúng ta nói rằng biến iPtr chờ "trỏ đến" vị địa chỉ bộ nhớ iValue thì hình sau đây minh họa điều này.

Biến pointer trỏ đến địa chỉ của byte đầu tiên trong vị trí bộ nhớ của biến nguyên iValue (được minh họa trong sơ đồ bằng cách chiếm giữ 4 byte liên tiếp).

Lúc toán tử sự tham chiếu được dùng với một biến pointer thì nó truy cập đến nội dung của biến mà nó trỏ đến, một mã sau đây sẽ minh họa cách dùng hai toán tử.

```
int x, y                    /* khai báo các biến x và y */
int * iPtr;                 /* trỏ đến một biến nguyên */
x=10;                       /* nội dung mới của biến x*/
iPtr = &x;                  /* biến iPtr trỏ đến biến x*/.
y=*iPtr;                    /* biến y được gán vào nội dung của x*/
```

*trong câu lệnh cuối cùng này *iPtr ám chỉ đến nội dung của biến x, giá trị của biến này = 10 được gán cho biến y.*

5.4.4. Hàm và các đối số của thủ tục sử dụng Pointer

Trong các ngôn ngữ chẳng hạn như C++ và C cơ cấu truyền bằng cách tham chiếu được thực thi qua việc sử dụng các pointer thì địa chỉ như

thế được truyền vào một chương trình con. Tự bản thân địa chỉ này không được chỉnh sửa, nhưng hàm được gọi và thủ tục được gọi địa chỉ đó. Bất cứ lúc nào địa chỉ được truyền dưới dạng tham số thực tế thì tham số hình thức tương ứng trong chương trình con được khai báo dưới dạng pointer.

VÍ DỤ 5.15 *Chương trình C sau đây mô tả giá trị của hai biến nguyên đã cho bằng cách sử dụng các pointer.*

```
#include <stdio.h>
void swap_values(int *, int *); /* function prototype */
int a=5, b=2; void main()
{
printf("\nThe initial values of the variables a and b are %d
  and %d respectively", a,b);
swap-values (&a,&b); /* swap content of variables */
printf('\n\nThe final values of the variables a and b are %d
  and %d respectively\n', a,b); } /* end of main */
void swap_values (int * iPtr1, int * iPtr2)
{
int temp;
temp=*iPtr1;
*iPtr1=*iPtr2;
*iPtr2=temp;}
```


Lưu ý rằng địa chỉ của các tham số thực tế được truyền sang thủ tục con bằng cách sử dụng toán tử khử tham chiếu và rằng các tham số hình thức tương ứng được xác định dưới dạng các pointer.

Trong nguyên mẫu hàm void swap_value (int, int*); kiểu dữ liệu của hai tham chiếu được xác định dưới dạng "pointer to intergers". Không có sự đề cập đến tên của tham số, kiểu nguyên mẫu này có thể được xác định như minh họa dưới đây mà không làm ảnh hưởng đến việc thực thi chương trình.*

*void swap_value (int * x, int * y)*

Như chúng ta đã chỉ định trước đây, vai trò của biến x và y trong nguyên mẫu đó là các chức năng giữ chỗ (placeholders). Không có tham số hình thức tham số thực tế nào được gọi tên sau bất kỳ các biến nào ở đây.

VÍ DỤ 5.16 *Chương trình sau đây minh họa hàm từ ví dụ 5.15 trên C++. Nó cho thấy một cách khác nếu truyền đối số bộ tham chiếu. Nó sử dụng toán tử & hoặc toán tử địa chỉ thay vì sử dụng *pointer.*

```
void swap_values (int& iPtr1, int& iPtr2)
{ int temp;
temp=iPtr1;
iPtr1=iPtr2;
iPtr2=temp;}
```

Lưu ý rằng các tham số hình thức C++ sử dụng & để chỉ ra rằng vị trí bộ nhớ đang truyền từ chương trình gọi. Các biến được gửi dưới dạng các tham số thực tế sẽ bị thay đổi. Dòng cần thiết để gọi hàm số này thì không gửi các pointer nhưng lại gửi từ bản thân các biến.

```
swap_values (a,b); /* swap content of variables */
```

Trong Visual Basic các cơ cấu tham số mặc định chính là truyền theo tham chiếu. Ví dụ sau đây minh họa điều này.

***VÍ DỤ 5.17** Thủ tục sau đây thay đổi các ký tự kiểu chữ in thường ASCII sang ký tự kiểu chữ in hoa.*

Giả sử rằng có một texbox có tên là txtUserInput. Thủ tục con liên kết với biến cố KeyPress được minh họa dưới đây. Như tên đã ngụ ý, biến cố này chỉ xảy ra lúc người dùng nhấn một phím có mã ASCII tương ứng.

```
Private Sub txtuserInput_KeyPress (KeyAscii As Integer)
Dim Character as String
Character=Chr(KeyAscii)          'convert input value to character
Character=Ucase(Character) 'convert character to uppercase
KeyAscii=Asc(Character) 'convert character to ASCII code
End Sub
```


Giá trị đầu vào của thủ tục này chính là biến nguyên Keyascii. Biến này biểu thị mã ASCII của phím được người dùng nhấn.

Đầu biến đổi giá trị nguyên ký tự bằng cách sử dụng hàm được tạo Chr(). Ký tự này sau đó được biến đổi sang một ký tự kiểu chữ in hoa bằng cách sử dụng hàm được tạo sẵn Ucase. Sau cùng ký tự chữ in hoa được biến đổi sang mã ASCII tương đương của nó bằng cách sử dụng hàm được tạo sẵn Asc(). Mặc dù chúng ta đã sử dụng ba câu lệnh riêng biệt để thực thi các phép biến đổi này, nhưng vẫn có thể kết hợp tất cả ba hàm trong một câu lệnh đơn.

```
KeyAscii = Asc(Ucase(Chr(KeyAscii)))
```

Lưu ý rằng các thay đổi cho tham số đầu vào được phản ánh ngay lập tức bởi vì các tham số thực tế và tham số hình thức đều đang tham chiếu đến cùng một địa chỉ.

5.4 .5 Truyền tham số trong Java

Java không phải là một ngôn ngữ thủ tục y như C hoặc C++. Trong Java mỗi hàm hoặc mỗi phương pháp đều là một phần của một lớp lớn hơn. Xem chương 8 để biết giải thích chi tiết về cấu trúc lớp ngôn ngữ Java. Các hàm Java có hình thức và đặc tính y hệt như các hàm trong C++. Ngoại trừ rằng tất cả các đối tượng lớp đều được truyền bởi tham chiếu và tất cả các kiểu đơn giản được truyền theo giá trị. không có dấu ampersand (&) cũng không có dấu asterisk (*) được dùng. Chỉ có các hàm Java đơn giản đã được minh họa ở đây còn mã gọi lớp thì không.

***VÍ DỤ 5.18** Hãy viết một hàm Java để trả về một giá trị gấp đôi biểu thị cho diện tích của vòng tròn, cho biết bán kính nguyên là một tham số.*

Sử dụng 3.14 làm giá trị của p với diện tích của hình tròn là p bán kính bình phương. hàm số như dưới đây:*

```
public double FindAreaCircle(int radius)
{
  final double PI=3.14;
  return (PI *radius*radius);
}
```

SOLVED PROBLEMS
Bài tập có lời giải

5.1 The narrative shown below describes the tasks of a subprogram. Is this an appropriate description for a function? If it is not, can it be an appropriate description for a subprocedure?

"The subprogram reads a sequence of integers typed by the user and returns the average, maximum, and minimum values of the sequence."

This is not an appropriate description for a function. A function can return only a single value and this narrative calls for a subprogram that returns three possible different values.

This is not an appropriate description for a subprocedure either. Subprocedures and functions should perform single tasks. The narrative describes four individual tasks to be carried out by a single subprogram. Each of these actions should be carried out by an individual subprogram. The sequence of values should be read by a subprocedure. The average, the maximum, and the minimum should be calculated by three individual functions.

5.2 The following narrative describes the tasks of a subprogram. What type of subprogram should perform these actions?

"The subprogram reads a sequence of characters into a series of consecutive memory locations and eliminates leading (blanks at the front) and trailing blanks (blanks at the back). It also eliminates a sequence of one or more intervening blanks (blanks in the middle) by a single blank."

Functions and subprocedures should perform actions that can be described by a sentence containing a single subject, a single verb, and a single object. Following this convention, it is better to subdivide the task called for by this narrative into several subtasks. Each subtask, in turn, should be performed by a single subprogram. Therefore, there should be a single subprocedure to perform each of the following activities: read the characters into the consecutive memory locations, eliminate leading blanks, eliminate intervening blanks, and eliminate trailing blanks. From a design point of view, each of these procedures should be considered as a building block. Once these subprocedures have been written, they can be used in subsequent programs to perform similar tasks.

5.3 Write the function prototype for the C functions described below.

(a) Function name: random number generator
Return data type: integer
Input data types: none

(b) Function name: greatest common divisor
Return data type: integer
Input data types: two integer values

(c) Function name: delete leading blanks
Return data type: none
Input data types: none

(d) Function name: string concatenation
Return data type: void
Input data type: pointer to a character, a pointer to a character constant

Names of identifiers in the C language are written with no intervening blanks. We will use a combination of upperand lowercase to write the function names.

(a) int RandomNumberGenerator (void);

Since the function returns an integer we can use the keyword int to specify the return data type. The keyword void indicates that no input parameters are expected.

(b) int GreatestCommonDivisor (int, int);

This function returns an integer value, therefore, we can use the keyword int to specify the return data type. There are two integer inputs, therefore, we need to write the keyword int twice separated by a comma. There is no need to write the name of any identifier in the function prototype; however, this prototype could have been written as

```
int GreatestCommonDivisor (j int, k int);
```

None of the identifiers in the function header needs to be called either j or k.

(c) void DeleteLeadingBlanks (void);

This function does not return any value nor receive any input. Therefore, we need to use the keyword void to indicate that no returning value should be expected from this function. Likewise, we indicate that no input parameter should be expected using the keyword void.

(d) void StringConcatenation (char * , const char *);

The keyword void indicates that the function does not return any value. The first parameter to this function is a pointer to a

character variable or string. The second parameter is a pointer to a character constant.

5.4 Write a Visual Basic program that prompts the user for an input string and translates each of the characters of the input sequence into Morse code using the equivalence table given below. Use a function that returns the Morse equivalent of each letter.

Morse Code

A	. _	J	. _ _ _	S	. . .
B	_ . . .	K	_ . _	T	_
C	_ . _ .	L	. _ . .	U	. . _
D	_ . .	M	_ _	V	. . . _
E	.	N	_ .	W	. _ _
F	. . _ .	O	_ _ _	X	_ . . _
G	_ _ .	P	. _ _ .	Y	_ . _ _
H	. . . .	Q	_ _ . _	Z	_ _ . .
I	. .	R	. _ .		

Assume that there is only one form called frmMorse with two textboxes called txtPlainText and txtMorseCode. To limit the length of the output Morse code to less than 80 characters, limit your input line to 30 characters maximum.

The code for this program is:

```
Option Explicit
Private Sub txtPlainText_KeyPress(KeyAscii As Integer)

 Dim Character As String

  Character=txtMorseCode.Text & " " & EnglishToMorse (Chr(KeyAscii))
  txtMorseCode.Text=Character
End Sub
```

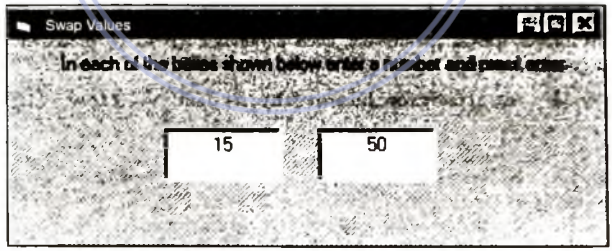
```
Private Function EnglishToMorse(stInputCharacter As String) As String

Select Case stInputCharacter
 Case Is="A", "a"
   EnglishToMorse=". _"
 Case Is="B", "b"
   EnglishToMorse="_ . . ."
 Case Is="C", "c"
   EnglishToMorse="_ . _ ."
 Case Is="D", "d"
   EnglishToMorse="_ . ."
 Case Is="E", "e"
   EnglishToMorse="."
 Case Is="F", "f"
   EnglishToMorse=". . _ ."
 Case Is="G", "g"
   EnglishToMorse="_ _ ."
 Case Is="H", "h"
   EnglishToMorse=". . . ."
 Case Is="I", "i"
   EnglishToMorse=". ."
Case Is="J", "j"
  EnglishToMorse=". _ _ _"
Case Is="K", "k"
  EnglishToMorse="_ . _"
Case Is="L", "l"
  EnglishToMorse=". _ . ."
Case Is="M", "m"
  EnglishToMorse="_ _"
Case Is="N", "n"
  EnglishToMorse="_ ."
Case Is="O", "o"
  EnglishToMorse="_ _ _"
Case Is="P", "p"
  EnglishToMorse=". _ _ ."
Case Is="Q", "q"
  EnglishToMorse="_ _ . _"
Case Is="R", "r"
  EnglishToMorse=". _ ."
Case Is="S", "s"
  EnglishToMorse=". . ."
Case Is="T", "t"
  EnglishToMorse="_"
Case Is="U", "u"
  EnglishToMorse=". . _"
```

```
Case Is="V", "v"
  EnglishToMorse="· · · _"
Case Is="W", "w"
  EnglishToMorse="· _ _"
Case Is="X", "x"
  EnglishToMorse="_ · · _"
Case Is="Y", "y"
  EnglishToMorse="_ · _ _"
Case Is="Z", "z"
  EnglishToMorse="_ _ · ·"
Case Else
  EnglishToMorse=""
End Select

End Function
```

The function EnglishToMorse translates, character by character, the input typed by the user. The function EnglishToMorse receives as its input the ASCII character code of keys pressed by the user. In all other cases the function generates a null character. The Select Case statement allows the function to choose the appropriate Morse code for any letter of the English alphabet. Notice that the output characters are separated by a blank.

5.5 Implement the swap algorithm in Visual Basic.

The initial input screen of this program may look like this:



The code to implement this program is shown below:

```
Option Explicit
    Private Sub TxtVal1_KeyPress(KeyAscii As Integer)
    Dim Character As String

    If KeyAscii=vbKeyReturn Then
      If IsNumeric(txtVal1.Text) Then
        txtVal2.SetFocus
      Else
        MsgBox "Enter a numeric value", vbInformation, "Invalid Input"
        txtVal1.SetFocus
```

```
        End If
      End If
    End Sub
      Private Sub TxtVal2_KeyPress(KeyAscii As Integer)
      Dim Character As String
      If KeyAscii=vbKeyReturn Then
        If IsNumeric(txtVal2.Text) Then
          SwapValues txtVal1, txtVal2
        Else
          MsgBox "Enter a numeric value", vbInformation, "Invalid Input"
          txtVal1.SetFocus
        End If
      End If
      Private Sub SwapValues(iBox1 As TextBox, iBox2 As TextBox)
      Dim temp As String
        temp=iBox1.Text
        iBox1.Text=iBox2.Text
        iBox2.Text=temp
End Sub
```

Observe that the code associated with each of the textboxes verifies that the user has typed a numeric value by means of the built-in function Val. If the value typed by the user is not numeric, the program warns the user of this fact by means of a message box, and the focus remains in the textbox. This program assumes that the user proceeds in an orderly fashion and it never attempts to start typing in the second textbox.

5.6 What is wrong with the following header declaration in C?

```
int function_header(int x, y, float z)
```

This is an erroneous declaration since each parameter needs to be declared separately. If the user wants to declare two integer formal parameters, each parameter has to be preceded by the qualifier int. Therefore, the correct format is

```
int function_header(int x, int y, float z)
```

5.7 If val1, val2 and val3 are integer variables, is it incorrect to invoke the function of Example 5.1 as max_of_three val1,val2, val3?

Calling the function max_of_three integers as indicated above will produce an error since the parameters to the function need to be passed enclosed in parentheses.

5.8 What is wrong with the following C function?

```
//function to calculate the maximum of two given
  integer values
int maximum_of_two (int x, int y)
{
```

```
int temp ;
     if    ( x>y )
           temp=x;
     if    (y>x)
           temp=y;
}
```

There are two types of error in this function. The first is syntactical, since the function does not return a value. This can be corrected by writing

return temp

before the closing curly bracket. In this particular function there is no need to use a temporary variable. We could have used **return x** or **return y** to simplify the code.

The second type of error is a little bit more difficult to discover. Notice that the function does not consider the case that the two input variables may have the same value. In other words, assuming that the function is syntactically correct, the function will return the maximum of the two input variables only when these variables are different. To correct this error one of the two if conditions needs to be changed so it tests for values that are "greater or equal." Therefore, the function can be written as follows:

```
int maximum_of_two (int x, int y)
{
  if ( x>=y)
    return x;

  if ( y>x)
     return y;
}
```

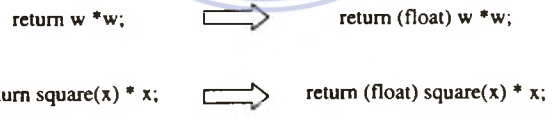
5.9 Are there any errors in the following function declarations and function calls? If there are, correct them. After correcting the program, try to run it using a C compiler. Are there any warnings? If there are, correct them.

```
//Function prototypes
float square (int x);
float triple_it (int y);
//Function calls
```

```
dval1=square ( value1).;   // assume that value1 is
  an integer variable and dval1 a double variable
dval2=triple_it (value1); // dval2 is a double vari-
  able
//Function declarations
void square ( int w)
{
  return w *w;
}
float triple_it (int x)
{
  return square (x) * x;
}
```

The header of the square function has been declared with the void qualifier whereas its prototype has been declared as returning a float. Correct this error by changing the function header to float.

If we run this program after correcting it, we will see that the C compiler produces a warning about a possible loss of data during conversion. This refers to the fact that the input parameters are integers but the function returns a float. To correct this, short of changing the type of the input parameters, we can use a cast operator to convert the results of the operations before returning any value to the calling program. To do this, change the return statements as indicated below.



Notice that in the function triple_it we have used the square function as if it were any other variable.

5.10 Assume that the following declarations appear in a C++ program. Are these declarations illegal? Can these two declarations appear in the same program?

No. These declarations are not illegal. C++ allows the declarations of two or more functions with the same name and in the same program provided that these functions have different sets of parameters. This means that if the parameters are all of the same type then the number of parameters in both functions must be different. If both functions have the same number of parameters then, in one of the functions, at least one of the parameters must be of different type. This is known as **function overloading.** This feature of C++ and some other languages allows the programmer to operate on different data types using functions with identical names. The combination of function name and number of parameters is known as the **signature of the function.** Using this terminology, we can say that C++ allows function overloading provided that the signatures of the functions are different.

5.11 Write function prototypes in C to match the following descriptions.

(a) a function named ValidateNumber has three arguments. The first argument is an integer, the second argument is a float number and the third argument is a double. The function returns a float.

(b) a function named ValidRange operates on three global variables and returns a float value in the range of -1 to 1 if the number is in the valid range.

(c) a function PowerOfTwo accepts a floating-point argument and returns the result of raising the argument to the second power.

(a) Since the ValidateNumber function returns a float value and has three arguments, its skeleton looks like this:

```
float ValidateNumber( x,y,z);
```

Replacing the placeholders x, y, and z with their corresponding data types, the function prototype is

```
float ValidateNumber (int, float, double);
```

We can also write this prototype as float ValidateNumber(int x, float y, double z);.

(b) Since the function ValidRange operates on three global variables, it does not need any arguments. Knowing that the function returns a float value we can write its prototype as

```
float ValidRange (void)
```

The fact that the function returns a value between -1 and 1 cannot be expressed as part of the prototype. It is the responsibility of the programmer to ensure that the function meets this requirement.

(c) Since the function PowerOflivo returns a float value and accepts a single argument, its skeleton is float(x). Replacing the place-

holder x with the data type of the input parameter, the prototype for this function is

```
float PowerOfTwo (float);
```

This function prototype can also be written as

```
float PowerOfTwo (x float);
```

5.12 Write function headers in Visual Basic to match the descriptions shown below. Assume that all functions are private.

(a) A function named factorial accepts an integer parameter and returns an integer.

(b) A function named ValidRange accepts as input a double value and returns True or False depending on whether or not the value is within a given range.

(c) A function named ConvertToNumeric accepts as input a text string representing an integer value and returns its numerical equivalent.

(a) As indicated before, the data type for functions and variables in Visual Basic is indicated using the As Data type clause. According to this, the header of the factorial function may look like this

```
Private Function Factorial (iValue as Integer) As Integer
```

(b) Since the function ValidRange returns true or false, its data type must be Boolean. Therefore, the function header may look like this:

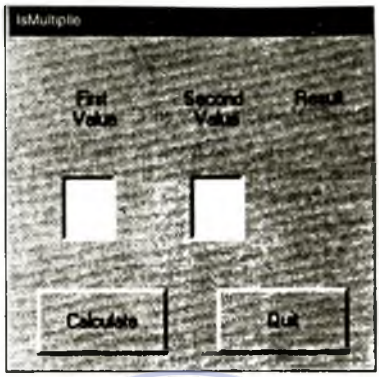
```
Private Function ValidRange (dValue As Double)
    As Boolean
```

(c) The function ConvertToNumeric accepts as input the text string representation of an integer and returns its numerical representation. Therefore, the function header may look like this:

```
Private Function ConvertToNumeric (stInputValue as
    String) As Integer
```

5.13 Write a Visual Basic program that accepts two integer values from the user and determines whether the first input value is multiple of the second. Use a function named IsMultiple that returns true or false as appropriate.

We could use a simple form with two textboxes to accept the user inputs. The result (True or False) can be displayed on the form by assigning it to the caption of a label on the form. In addition, we may have a command button to do the calculations and another one to end the program. The form may look like this:



The code associated with the command button Calculate is shown below.

```
Private Sub Calculate_Click()
   Dim i As Integer
   Dim j As Integer
i=Val(txtVal1)
j=Val(txtVal2)
If ismultiple(i, j) Then
   lblMultiple.Caption=True
Else
   lblMultiple.Caption=False
End If
End Sub
```


The values of the textboxes are initially converted to their numerical equivalents using the built-in function Val. Although the IsMultiple function could have been called as follows: IsMultiple(Val(txtVal1),Val(txtVal2)), it is better to transform the contents of the textboxes outside the function call to make the code a little bit more readable.

The code associated with the Quit function is very simple since it consists of a single statement, the End statement, which, as its name indicates, ends the execution of the program:

```
Private Sub Quit_Click()
   End
End Sub
```

The function IsMultiple uses the Mod operator to determine whether the first argument is a multiple of the second argument. This operator returns the remainder of the division of the first argument by the second argument. Notice that we compare the result of the Mod operation to zero since it is a mathematical fact that a number is a multiple of another if the remainder of dividing the first number by the second is equal to zero.

```
Private Function Ismultiple(iVal1 As Integer, ival2 As
   Integer) As Boolean
If (iVal1 Mod ival2)=0 Then
   IsMultiple=True
Else
   Ismultiple=False
End If
End Function
```

5.14 Write the function of the previous exercise in C++.

In the C family, including C++, there is no Boolean data type as defined in Visual Basic. However, we will consider that the function returns a true value if it returns a 1 and false if it returns a zero. With this in mind, we can write the program as follows:

```
#include <iostream.h> int IsMultiple(int, int); //func-
   tion prototype
void main ()
{ int i,j;
   cout<< "\nEnter the first integer value:";
   cin>>i; cout<< "\nEnter the second integer value:";
   cin>>j ;
if (IsMultiple(i,j)==0)
   cout<< "\nThe first value is multiple of the
   second\n\n";
else
   cout<< "\nThe first value is not multiple of the
   second\n\n.";}
int IsMultiple(int x, int y)
{
   if (x % y==0)
      return 1;
   else
return 0;
```

Notice that the C++ operator % is equivalent to the Mod operator of Visual Basic.

5.15 Write a function in Visual Basic that accepts as input a value between zero and 100 and returns a letter grade according to the grading scale shown below.

Numerical Grade	Letter Grade
90 through 100	A
80 through 89	B
70 through 79	C
60 through 69	D
less than 60	F

We will assume that the function is private and part of a much larger program not shown here.

```
Private Function LetterGrade(Grade As Double) As String
   Select Case Grade
   Case 90 To 100
       LetterGrade="A"
   Case 80 To 89
       LetterGrade="B"
   Case 70 To 79
       LetterGrade="C"
   Case 60 To 69
       LetterGrade="D"
   Case 0 To 59
       LetterGrade="F"
   End Select
End Function
```


In this case we use the Select statement because it allows us to test for ranges of values in a more compact and more understandable way than a set of nested if statements.

5.16 Using the program skeleton shown below, determine the scope of all declared variables in C.

```
int price_per-unit;
float interest-rate;
double earnings;
void main()
```

```
{ int stock-type;
   float commission;
   int rating;
   :
   :
}
double depreciation (int val1, int val2)
{ double index_deval;
float earnings;
   :
   :
return(index-deval);
int internal-revs (float val1, float val2)
{ int rating;
float commission;
   :
   :
return(interest_rate*(rating-(val1/val2)));
}
```


Variables price_per_unit and interest rate are both global variables that can be referenced by any of the functions of this program.

The global variable earnings can be referenced by all functions except by the depreciation function. Notice that this function has a local variable by the same name.

Variables stock_type, commission, and rating are all local to main. No other function can reference these variables.

Variables index deval, earnings, val1, and val2 are local variables to the depreciation function. No other function can reference these variables.

Variables rating, commission, val1, and val12 are local variables to the internal revs function. No other function can reference these variables.

Notice that the formal parameters val1 and val2 are known only within this procedure and do not have anything in common with the formal parameters of the same name in the depreciation function. Similar comment can be made with respect to the variables rating and commission in this function and the variables by the same names in the main function.

5.17 What is the scope of the following variables in the program skeleton shown below? What global variables can be referenced in both main and function1?

```
int a, b;
char c, d;
void main()
{  float a, d;
   :
}
int function1 (int a)
{  char d;
   int b;
   :
}
```

Only variable c can be referenced simultaneously by main and function1. Notice that in main there are two local variables named a and d. When these variables are referenced in main they always refer to the local variables, not the global variables. The global variable b can be referenced inside main since there is no local variable with this name.

Similarly, function1 has three local variables named a, b, and d. Within this function, any reference to any variable with any of these names will always refer to the local variables and not the global variables. Since none of the local variables of function1 is called c, we can always refer to this global variable.

5.18 In the programming languages of the C family, if value1 is an integer variable, what is the meaning of &value1?

value1 is a named memory location. We refer to the content of this memory location whenever we reference value1. &value1 is the address of the memory location value1. As Example 5.14 illustrates, there is a difference between the content of a variable and the address of such a variable

5.19 Write the appropriate C statements that correspond to the following situations.

(a) iptr is a pointer to an integer variable.

(b) the variable pointed to by iptr is an integer.

(c) the variable pointed to by cptr is a character variable.

(d) the integer variable whose address is in value_ptr.

(a) and (b) refer to the same situation although they are phrased a little bit differently int *iptr declares iptr as a pointer variable to an integer.

(c) char *cptr declares that cptr is a pointer to a character variable. In other words, the variable pointed to by cptr is a character variable.

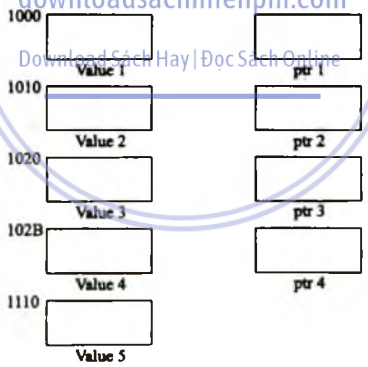
(d) If value_ptr is a pointer to an integer variable and it contains already the address of a particular

variable, we can always refer to the variable as *value_ptr. This can be better illustrated as

follows:

```
int j, *value_ptr;
value_ptr=&j;
*value_ptr=*value_ptr+1; // the content of variable j
                            is incremented by 1.
```

5.20 Consider the following set of memory locations and C-like instructions. What is the content of the variables and pointers after executing the instructions shown below in sequence? Assume that all variables are of the same type and that the Ptr variables are pointers to that type of variables.



(a) value3 = 15

(b) value4 = 12

(c) Ptr3 = &value4

(d) Ptr1 = &value1

(e) value1 = *Ptr3 + 2

(f) Ptr2 = &value3

(g) value2 = *Ptr2 + *Ptr3

(h) Ptr4 = &value2

(i) value5 = *Ptr1 +*Ptr2+*Ptr3*Ptr4

Instructions (a) and (b) set the content of variables value3 and value4 to 15 and 12 respectively.

Instructions (c) and (d) copy the addresses of variables value4 and value1 into the pointer variables Ptr3 and Ptr1 respectively. According to the diagram, the content of Ptr1 is now 1000 and the content of Ptr3 is 102B.

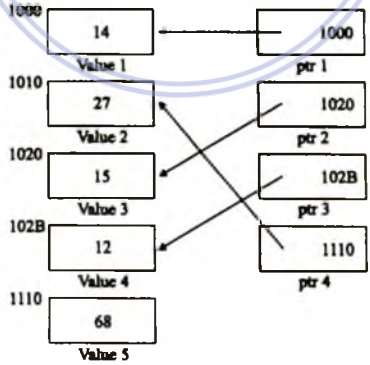
Instruction (e) sets the content of variable value1 to the contents of variable value4 plus 2. Notice that *Ptr3 is the content of the variable whose address is currently stored in Ptr3. Observe that the current content of Ptr3 is 102B. This is the address of variable value4. *Ptr3 is the content of the variable value4, which, according to the diagram, is 12. Therefore, the content of variable value1 is 12 + 2.

Instruction (f) copies the address of variable value3 into Ptr2. The new value of Ptr2 is 1020.

Instruction (g) adds the contents of the variables whose addresses are in Ptr2 and Ptr3. Since the content of variable value3 (address 1020) is 15 and the content of variable value4 (address 102B) is 12, the content of variable value2 is 27.

Instruction (h) copies the address of variable value2 into Ptr4.

Instruction (i) adds the content of variables value1, value2, value3, and value4 and stores the result into variable values. The new content of variable values is 68.



5.21 Write a C++ subprocedure to implement Solved Problem 5.15. Write the subprocedure with two parameters; the first parameter is a numeric input parameter between 0 and 100. The second parameter is used to return the corresponding letter grade. Pass the first parameter by value and the second parameter by reference.

A subprocedure to accomplish this task is shown below. Notice that the first input parameter is a float variable (passed by value) and the second parameter is a pointer to a character variable (passed by reference). Therefore, when the subprocedure is called the address of the character variable grade needs to be passed as an actual parameter. In this procedure a set of nested if . . . else if statements is used instead of the case statement of problem 5.15 because the switch statement in C++ does not allow us to test for a range of values.

```
#include<iostream.h>
void GetLetterGrade(float, char*);
float score;
char grade;
void main()
{ cout<<"\nPlease enter a score between 0 and 100:;
cin>> score;
GetLetterGrade(score,&grade);
cout<<"\nThe letter grade is "<<grade<<endl<<endl; }
/*This procedure receives as input a value between 0 and
   100 and "returns" a letter grade corresponding to the
   grade. The first parameter is passed by value; the sec-
   ond parameter is passed by reference. Ptr is a pointer
   to a character variable.*/
void GetLetterGrade(float i, char* ptr)
{
if (i>=90 && i<=100)
   *ptr='A';
else if (i>=80 && i<=89)
   *ptr='B';
else if (i>=70 && i<=79)
   *ptr='C';
else if (i>=60 && i<=69)
   *ptr='D';
else if (i>=0 && i<=59)
   *ptr='F';
}
```

5.22 Write a C++ subprocedure that accepts as input a number of nickels, dimes, and quarters and returns the corresponding dollar amount. Pass the variable corresponding to the dollar amount by reference.

```
#include<iostream.h>
void HowManyDollars(int,int,int,float&);
void main()
{ float dollar_amount;
int nickels, dimes, quarters;
cout<< "\nPlease enter number of nickels:";
cin>>nickels;
cout<< "\nPlease enter number of dimes:";
cin>> dimes ;
cout<< "\nPlease enter number of quarters:";
cin>> quarters;
HowManyDollars(nickels,dimes,quarters,dollar_amount);
cout<<"\nThe total amount in dollars is
   "<<dollar_amount<<endl<<endl;
}
void HowManyDollars(int n, int d, int q, float& t)
{
   t=(float)(n*0.05+d*0.10+q*0.25);
}
```

Notice that we have used the cast operator to convert the entire result to afloat value. C++ converts the integer values to double before doing the individual multiplication operations. The resulting addition is of type double. If we do not use the cast operator the compiler will produce a warning.

5.23 Write a Java function to return the double value of miles per gallon, given the integer distance and the integer gallons used as a parameter.

```
public double FindMPG (int distance, int gallonsUsed)
{
   return (double) distance/gallonsUsed;
}
```

5.24 Write a Java method that implements the problem in Solved Problem 5.22. The function should accept a number of nickels, dimes, and quarters and return the corresponding dollar amount. Remember that Java can only return a value directly using the return function.

```
public float HowManyDollars (int nick, int dime, int quart)
{
   return (float)(nick*0.05+dime*0.10+quart*.25);
}
```

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

BÀI TẬP CÓ LỜI GIẢI

5.1 *Phần minh họa dưới đây mô tả các tác vụ của chương trình con. Có phải là đây một phần mô tả phù hợp cho hàm hay không? Nếu không thì nó có thể là một phần mô tả phù hợp cho một thủ tục con không?*

"Chương trình con đọc một chuỗi các số nguyên được người dùng gõ nhập rồi trả về giá trị trung bình, giá trị cực đại và giá trị cực tiểu của dãy".

Đây không phải là phần mô tả phù hợp dùng cho một hàm. Một hàm chỉ có thể trả về một giá trị đơn và phần trên đây để gọi một chương trình con để trả về ba giá trị khác nhau có thể có.

Đây cũng không phải là phần mô tả phù hợp dành cho một thủ tục con. Các thủ tục con và các hàm chỉ thực hiện những tác vụ đơn giản. Phần nêu trên chỉ mô tả 4 tác vụ riêng biệt được thực thi bởi một chương trình con đơn. Mỗi hành động này nên thực thi bởi một chương trình con riêng biệt. Dãy của các giá trị được đọc bởi thủ tục con, giá trị trung bình, giá trị cực đại, giá trị cực tiểu sẽ được tính bởi ba hàm riêng biệt này.

5.2 *Phần sau đây mô tả các tác vụ của một chương trình con. Kiểu chương trình con thực thi những hành động này là gì?*

"Chương trình con đọc một dãy các ký tự thành một chuỗi các vị trí bộ nhớ nằm kề nhau và loại bỏ phần đầu (phần trống đầu tiên) và phần trống đuôi (phần trống phía sau)" nó cũng loại bỏ một hoặc nhiều dãy ở giữa (các khoảng trống ở giữa) theo một dãy các hàm và các thủ tục con sẽ thực thi các hành động có thể được mô tả bởi một câu có chứa một đối tượng, một chủ ngữ, một động từ và một túc từ. Cho phép biến đổi sau đây tốt hơn bạn chia nhỏ tác vụ được gọi bởi câu trên đây thành tác vụ con. mỗi tác vụ con lần lượt được thực thi bởi một chương trình con. Do đó sẽ có một thủ tục để thực thi mỗi một trong số các hành động sau đây: đọc các ký tự thành các vị trí bộ nhớ liên tục, loại bỏ các khoảng trống đứng đầu, các khoảng trống ở giữa và các khoảng trống đuôi. Từ quan điểm của người thiết kế mỗi trong số các thủ tục này nên được xem như một khối cấu trúc. Một khi những thủ tục này đã được viết, chúng có thể được dùng trong các chương trình kết quả để thực thi những tác vụ giống nhau.

5.3 *Hãy viết nguyên mẫu hàm dành cho các hàm C được mô tả dưới đây:*

(a) Tên hàm: random number generator

Kiểu dữ liệu trả về: integer

Kiểu dữ liệu nhập về: none

(b) Tên hàm: gretest common divisor

Kiểu dữ liệu trả về: integer

Kiểu dữ liệu nhập về: two integer value

(c) Tên hàm: delete leading blanks

Kiểu dữ liệu trả về: none

Kiểu dữ liệu nhập về: none

(d) Tên hàm: string concatenation

Kiểu dữ liệu trả về: void

Kiểu dữ liệu nhập về: trỏ đến một ký tự, một con trỏ đến một hằng số ký tự.

Các tên của bộ nhận dạng trong ngôn ngữ C được viết mà không có khoảng trống ở giữa. Chúng ta sẽ sử dụng một tổ hợp các kiểu chữ in hoa và kiểu chữ thường để viết các tên hàm.

(a) int RandomNumberGenerator (void)

Bởi vì hàm này trả về một giá trị số nguyên nên chúng ta có thể sử dụng từ khoá int để chỉ định kiểu dữ liệu trả về. Từ khoá void chỉ định rằng không có tham số đầu vào nào được mong đợi.

(b) int GreatestCommonDivisor (int,int)

Hàm này trả về một giá trị nguyên do đó ta có thể sử dụng từ khóa int để chỉ định kiểu dữ liệu trả về. Có hai đầu vào nguyên. Do đó ta cần phải biết từ khóa int hai lần được tách rời nhau bằng một dấu phẩy. Không cần phải viết tên của bất cứ bộ định dạng nào trong nguyên mẫu hàm. Tuy nhiên, nguyên mẫu này có thể được viết dưới dạng

```
int GreatestCommonDivisor (j int, k int);
```

Không có phần nào trong các bộ định dạng trong tiêu đề của hàm cần phải được gọi j hoặc k.

(c) void DeleteLeadingBlanks (void)

Hàm này không trả về bất cứ giá trị nào và cũng không nhận bất cứ đầu vào nào. Do đó ta cần phải sử dụng từ khóa void để chỉ định rằng không có giá trị nào trả về được mong đợi từ hàm này. Cũng vậy chúng ta chỉ định rằng không có tham số chỉ định nào được mong đợi từ khoá void.

(d) void StringConcatenation (char*, cost chae*);

Từ khóa void chỉ ra rằng hàm số này không trả về bất cứ giá trị nào. Tham số đầu tiên cho hàm này là một pointer trỏ

đến một biến ký tự hoặc một chuỗi. Tham số thứ hai là một con trỏ, trỏ đến một hằng số ký tự.

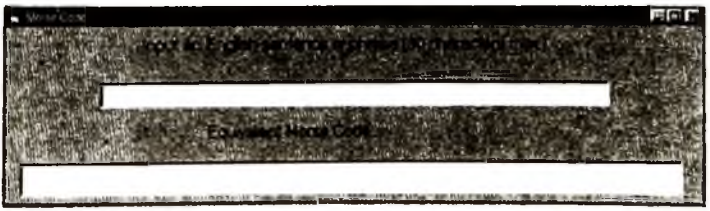
5.4 *Hãy viết một chương trình Visual Basic để nhắc nhở người dùng về một chuỗi nhập. Nó diễn dịch mỗi một trong số các ký tự của dãy nhập vào thành mã Morse bằng cách sử dụng một bản tương đương được cho dưới đây.*

Sử dụng một hàm để trả về tương đương Morse của một ký tự.

Mã Morse

A	. _	**J**	. _ _ _	**S**	. . .
B	_ . . .	**K**	_ . _	**T**	_
C	_ . _ .	**L**	. _ . .	**U**	. . _
D	_ . .	**M**	_ _	**V**	. . . _
E	.	**N**	_ .	**W**	. _ _
F	. . _ .	**O**	_ _ _	**X**	_ . . _
G	_ _ .	**P**	. _ _ .	**Y**	. _ _ _
H	. . . .	**Q**	_ _ . _	**Z**	_ _ . .
I	. .	**R**	. _ .		

Giả sử rằng chỉ có một dạng được gọi là frmMorse với hai textboxes tên là txtPlainText và txtMorsecode. Để giới hạn chiều dài của mã Morse đầu ra thì phải nhỏ hơn 80 ký tự, giới hạn dòng đầu vào của bạn lên đến tối đa là 30 ký tự.



Mã dành cho chương trình này là:

```
Option Explicit
Private Sub txtPlainText_KeyPress(KeyAscii As Integer)
```

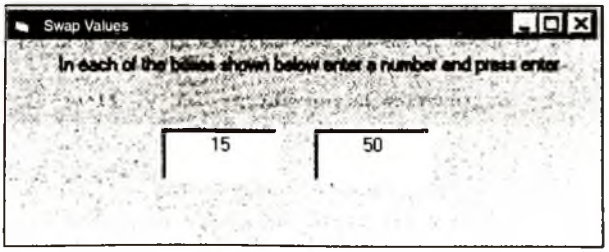
```
    Dim Character As String

     Character=txtMorseCode.Text & " " & EnglishToMorse (Chr(KeyAscii))
     txtMorseCode.Text=Character
    End Sub

    Private Function EnglishToMorse(stInputCharacter As String) As String

    Select Case stInputCharacter
     Case Is="A", "a"
       EnglishToMorse=". _"
     Case Is="B", "b"
       EnglishToMorse="_ . . ."
     Case Is="C", "c"
       EnglishToMorse="_ . _ ."
     Case Is="D", "d"
       EnglishToMorse="_ . ."
     Case Is="E", "e"
       EnglishToMorse="."
     Case Is="F", "f"
       EnglishToMorse=". . _ ."
     Case Is="G", "g"
       EnglishToMorse="_ _ ."
     Case Is="H", "h"
       EnglishToMorse=". . . ."
     Case Is="I", "i"
       EnglishToMorse=". ."
     Case Is="J", "j"
       EnglishToMorse=". _ _ _"
     Case Is="K", "k"
       EnglishToMorse="_ . _"
     Case Is="L", "l"
       EnglishToMorse=". _ . ."
     Case Is="M", "m"
       EnglishToMorse="_ _"
     Case Is="N", "n"
       EnglishToMorse="_ ."
     Case Is="O", "o"
       EnglishToMorse="_ _ _"
     Case Is="P", "p"
       EnglishToMorse=". _ _ ."
     Case Is="Q", "q"
       EnglishToMorse="_ _ . _"
     Case Is="R", "r"
       EnglishToMorse=". _ ."
     Case Is="S", "s"
       EnglishToMorse=". . ."
```

```
    Case Is="T", "t"
      EnglishToMorse="_"
    Case Is="U", "u"
      EnglishToMorse="· · _"
    Case Is="V", "v"
      EnglishToMorse="· · · _"
    Case Is="W", "w"
      EnglishToMorse="· _ _"
    Case Is="X", "x"
      EnglishToMorse="_ · · _"
    Case Is="Y", "y"
      EnglishToMorse="_ · _ _"
    Case Is="Z", "z"
      EnglishToMorse="_ _ · ·"
    Case Else
      EnglishToMorse=""
  End Select

End Function
```

Hàm EnglishToMorse diễn dịch theo từng giá trị vào mà người gõ nhập. Hàm số EnglishToMorse nhận dưới dạng dữ liệu vào của nó mã ký tự ASCII của các phím mà người dùng nhấn. Trong tất cả các trường hợp khác hàm này tạo ra một ký tự trống null. Câu lệnh Select Case cho phép hàm chọn mã Morse phù hợp cho bất cứ mẫu tự nào của bảng chữ cái tiếng Anh. Lưu ý rằng các ký tự xuất được tách với nhau bằng một khoảng trống.

5.5 *Thực thi thuật toán swap trong Visual Basic*

Màn hình đầu vào cấu tạo của chương trình này giống như dưới đây:



Mã thực thi chương trình này được cho sau đây:

```
Option Explicit
    Private Sub TxtVal1_KeyPress(KeyAscii As Integer)
    Dim Character As String

    If KeyAscii=vbKeyReturn Then
      If IsNumeric(txtVal1.Text) Then
        txtVal2.SetFocus
      Else
        MsgBox "Enter a numeric value", vbInformation, "Invalid Input"
        txtVal1.SetFocus
      End If
    End If
End Sub
    Private Sub TxtVal2_KeyPress(KeyAscii As Integer)
    Dim Character As String

    If KeyAscii=vbKeyReturn Then
      If IsNumeric(txtVal2.Text) Then
        SwapValues txtVal1, txtVal2
      Else
        MsgBox "Enter a numeric value", vbInformation, "Invalid Input"
        txtVal1.SetFocus
      End If
    End If

    Private Sub SwapValues(iBox1 As TextBox, iBox2 As TextBox)
    Dim temp As String

        temp=iBox1.Text
        iBox1.Text=iBox2.Text
        iBox2.Text=temp
End Sub
```


Quan sát cho thấy rằng mã được liên kết với mỗi một texboxes kiểm nghiệm rằng người dùng đã gõ nhập một trị số bằng cách dùng hàm tạo sẵn Val. Nếu giá trị được người dùng gõ nhập không phải là một con số thì chương trình cảnh báo cho người dùng sự kiện này bằng cách dùng một hộp thông điệp và các tiêu điểm còn giữ lại trong textbox, chương trình này giả sử rằng người dùng cần tiến hành theo một cách thức thứ tự không được thử bắt đầu gõ nhập trong texbox thứ hai.

5.6 *Có điều gì sai trong phần khai báo header trong C hay không?*

```
int function_header(int x, y, float z)
```

Đây là một khai báo lỗi bởi vì mỗi một tham số cần phải được khai báo riêng biệt. nếu người dùng muốn khai báo hai tham số hình thức nguyên, thì mỗi một tham số phải được đặt trước mỗi lượng từ in, do đó dạng đúng của nó là :

```
int function_header(int x, y, float z)
```

5.7 *Nếu Val1, Val2, Val3 là các biến nguyên, thì có gì sai sót khi ví dụ hàm 5.1 dưới dạng max_of_three val1, val2, val3?*

Bằng cách gọi hàm max_of_thre_integers như được chỉ định trên đây bạn sẽ tạo ra được một lỗi bởi vì các tham số cho hàm này cần phải được chuyển trong các dấu ngoặc đơn.

5.8 *Có gì sai với hàm C sau đây?*

```
//function to calculate the maximum of two given integer
  values
int maximum_of_two (int x, int y)
{
int temp ;
     if    ( x>y )
           temp=x;
     if    (y>x)
           temp=y;
}
```

Có hai kiểu lỗi trong hàm này. Thứ nhất là lỗi về cú pháp bởi vì hàm này không trả về một giá trị. Cho nên nó được chỉnh sửa bằng cách viết:

```
return temp
```

trước khi đóng các dấu móc ngoặc. Trong hàm đặc biệt này không cần phải sử dụng một biến tạm thời bạn có thể sử dụng return x hoặc return y để đơn giản hóa mã.

Kiểu lỗi thứ hai đó là kiểu lỗi khó khám phá. Lưu ý rằng hàm này không xem xét trường hợp ở đó hai biến đầu vào có giá trị giống nhau, nói cách khác thì giả sử thì hàm này giả sử đúng về mặt cú pháp thì hàm sẽ trả về giá trị cực đại hai biến đầu vào chỉ lúc những biến này khác nhau. Để chỉnh sửa lỗi này, một trong hai điều kiện x cần phải được thay đổi sao cho những thử nghiệm dành cho các giá trị là "lớn hơn hoặc bằng" do đó hàm có thể được viết như sau:

```
int maximum_of_two (int x, int y)
{
  if ( x>=y)
    return x;

  if ( y>x)
     return y;
}
```

5.9 Có lỗi nào trong phần khai báo hàm và cuộc gọi hàm sau đây hay không? Nếu có thể sửa chúng. Sau khi sửa chương trình hãy thử chạy nó bằng cách sử dụng bộ biên soạn C, có bất cứ phần cảnh báo nào hay không? Nếu có hãy chỉnh sửa nó.

```
//Function prototypes
float square (int x);
float triple_it (int y);
//Function calls
dval1=square ( value1).;       // assume that value1 is
  an integer variable and dval1 a double variable
dval2=triple_it (value1); // dval2 is a double variable
//Function declarations
void square ( int w)
{
  return w *w;
}
float triple_it (int x)
{
  return square (x) * x; )
}
```


Header của hàm bậc hai được khai báo với công cụ định lượng void trong khi nguyên mẫu của nó được khai báo dưới dạng một float. Hãy chỉnh sửa lỗi này bằng cách thay đổi tiêu đề hàm sang float.

Nếu chúng ta chạy chương trình này sau khi chỉnh sửa nó. Chúng ta sẽ thấy rằng trình biên soạn C tạo ra một cảnh báo về khả năng mất dữ liệu trong suốt quá trình biến đổi. Điều này ám chỉ đến sự kiện rằng các tham số đầu vào là những số nguyên nhưng hàm này là trả về một dấu chấm động. Để chỉnh sửa điều này hãy rút ngắn sự thay đổi kiểu tham số đầu vào. Ta có thể dùng một toán tử cast operator để biến đổi kết quả của các phép tính trước khi trả về bất cứ giá trị nào cho chương trình gọi. Để thực hiện điều này hãy thay đổi các câu lệnh trả về như được chỉ định dưới đây.

```
return w *w;          ⟹   return (float) w *w;

return square(x) * x; ⟹   return (float) square(x) * x;
```

Lưu ý rằng trong hàm triple_it chúng ta đã dùng hàm bậc hai y hệt như nó đã có trong bất kỳ biến nào khác.

5.10 *Giả sử rằng phần khai báo sau đây xuất hiện trong chương trình C++. Có phải chăng phần khai báo này không hợp lý? Có thể hai phần khai báo này xuất hiện trong cùng một chương trình được hay không?*

```
int triple_it (int x);
double triple_it(double x);
```

Các hàm khác nhau với các tên nhận dạng trong chương trình giống nhau

Không. Những khai báo này không hợp lệ. C++ cho phép các khai báo của hai hoặc nhiều hàm có tên giống nhau và trong cùng một chương trình giống nhau được cung cấp sao cho những hàm này có các tập hợp tham số khác nhau. Điều này có nghĩa rằng nếu các tham số tất cả có cùng kiểu giống nhau thì số các tham số trong hai hàm phải khác nhau. Nếu cả hai hàm có số tham số giống nhau thì ở trong một trong số hai hàm, phải có ít nhất một tham số có kiểu khác nhau. Đây được gọi là sự quá tải của hàm. Đặc tính này của C++ và một vài ngôn ngữ khác cho phép người lập trình phải thao tác trên các kiểu dữ liệu khác nhau bằng cách sử dụng các hàm với tên giống nhau. Tổ hợp các tên hàm và số các tham số được gọi là chữ ký của hàm. Bằng cách sử dụng thuật ngữ này chúng ta có thể bảo rằng C++ cho phép sự quá tải hàm được cung cấp để các chữ ký của hàm là khác nhau.

5.11 *Hãy viết các giao thức hàm trong C để kết hợp các phần mô tả sau đây.*

(a) Một hàm có tên là ValidateNumber có ba đối số. Đối số thứ nhất là một số nguyên, đối số thứ hai là số thập phân và đối số thứ ba là một số gấp đôi. Hàm này trả về một số thập phân float.

(b) Một hàm có tên là Validrange hoạt động trên ba biến tổng thể và trả về một giá trị động nằm trong miền từ trường 1 đến 1 nếu số này nằm trong phạm vi đúng.

(c) Một hàm PowerOfTwo chấp nhận một đối số dấu chấm động và trả về kết quả nâng đối số lên một lũy thừa bậc 2.

(a) Bởi vì hàm ValidateNumber trả về một giá trị động và có ba đối số cho nên khung của nó sẽ giống như dưới đây:

```
float ValidateNumber( x,y,z);
```

Thay thế các placeholders x, y và z với các kiểu dữ liệu tương ứng, nguyên mẫu hàm là

```
float ValidateNumber (int, float, double);
```

Chúng ta cũng có thể viết nguyên mẫu này dưới dạng float ValidateNumber (int x, float y double z).

(b) *Bởi vì hàm ValidRange hoạt động trên ba biến tổng thể, cho nên nó không cần bất kỳ đối số nào. Cần biết rằng hàm này trả về một giá trị động chúng ta có thể nguyên mẫu của nó dưới dạng*

```
float ValidRange (void)
```

Sự kiện hàm trả về một giá trị giữa -1 và 1 không thể được trình bày dưới dạng một phần của nguyên mẫu. đây chính là trách nhiệm của nhà lập trình để bảo đảm rằng chương trình đáp ứng được yêu cầu này.

(c) *Bởi vì hàm PowerOfTwo trả về một giá trị động và chấp nhận đối số đơn của nó cho nên khung của nó là float(x) thay thế placeholder x với kiểu dữ liệu với tham số nhập vào, nguyên mẫu của kiểu hàm này là:*

```
float PowerOfTwo (float);
```

Nguyên mẫu hàm này có thể tương ứng dưới dạng

```
float PowerOfTwo ( x float);
```

5.12 *Viết các header của hàm trong Visual Basic để kết hợp phần mô tả dưới đây. hãy bảo đảm rằng tất cả các hàm đều là riêng tư.*

(a) *Một hàm có tên là giai thừa chấp nhận một tham số nguyên rồi trả về một giá trị nguyên.*

(b) *Một hàm có tên là ValidRange chấp nhận một giá trị vào là một giá trị double và trả về True hoặc False phụ thuộc và trường hợp giá trị đó có nằm trong một miền đã cho hay không?*

(c) *Một hàm có tên là ConvertToNumeric chấp nhận dưới dạng một đầu vào một chuỗi text biểu thị một giá trị nguyên và trả về giá trị tương đương số của nó.*

(a) *Như được chỉ định trước đây kiểu dữ liệu của hàm và biến trong Visual Basic được chỉ định bằng các sử dụng mệnh đề kiểu As Data. Theo điều này thì header của hàm giai thừa như dưới đây:*

```
Private Function Factorial (iValue as Integer) As Integer
```

(b) *Bởi vì hàm ValidRange trở về true hoặc false, nên kiểu dữ liệu của nó phải là Boolean. do đó header của hàm sẽ giống như dưới đây:*

```
Private Function ValidRange (dValue As Double)
    As Boolean
```

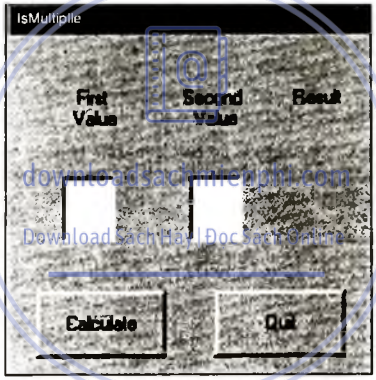
(c) *Hàm ConvertToNumeric chấp nhận đầu vào là cách trình bày của text của một số nguyên và trả về cách trình bày số*

của nó, dó đó tiêu đề hàm sẽ giống như ở dưới đây:

```
Private Function ConvertToNumeric (stInputValue as
                String) As Integer
```

5.13 *Hãy viết một chương trình Visual Basic chấp nhận hai giá trị nguyên từ người đùng và xác định cho biết giá trị nhập vào đầu tiên có phải là bộ số của giá trị thứ hai hay không. Sử dụng một hàm có tên là IsMultiple để trả về true hoặc false cho phù hợp.*

Chúng ta có thể sử dụng một form đơn giản có hai textbox để nhập giá trị nhập của người dùng. Kết quả (True hoặc False) có thể hiển thị trên form này bằng cách gán nó vào phần của thích của một nhãn trên form. Ngoài ra, ta có thể có một nút lệnh để thực hiện các phép tính và nút khác để kết thúc chương trình. Form này có thể có dạng như sau:



Mã có liên quan với nút Calculate được minh họa dưới đây:

```
Private Sub Calculate_Click()
   Dim i As Integer
   Dim j As Integer
i=Val(txtVal1)
j=Val(txtVal2)
If ismultiple(i, j) Then
   lblMultiple.Caption=True
Else
   lblMultiple.Caption=False
End If
End Sub
```

Giá trị của các textbox ban đầu được chuyển sang các số tương đương bằng cách sử dụng hàm được tạo sẵn là Val. Mặc dù hàm IsMultiple có thể được gọi như sau: IsMultiple(Val(txtVal1), Val(textVal2)), song tốt hơn là nên biến đổi nội dung của các textbox bên ngoài cuộc gọi hàm để làm cho mã dễ đọc hơn.

Mã liên quan với hàm Quit rất là đơn giản bởi vì nó chỉ có một câu lệnh duy nhất, câu lệnh End. Câu lệnh này kết thúc việc thực thi chương trình.

```
Private Sub Quit_Click()
   End
End Sub
```

Hàm IsMultiple sử dụng toán tử Mod để xác định xem đối số đầu tiên có phải là bội số của đối số thứ hai không. Toán tử này cho ra phần dư của phép chia: đối số thứ hai chia cho đối số thứ nhất. Lưu ý rằng chúng ta so sánh kết quả của toán tử Mod với số zero bởi vì trong toán học một con số là bội số của số khác nếu phần dư của phép chia số thứ hai cho số thứ nhất bằng zero.

```
Private Function Ismultiple(iVal1 As Integer, ival2 As
   Integer) As Boolean
If (iVal1 Mod ival2)=0 Then
   IsMultiple=True
Else
   Ismultiple=False
End If
End Function
```


5.14 *Viết một hàm dành cho bài tập trước đây trong C++.*

5.15 *Hãy viết một hàm trong Visual Basic để chấp nhận dưới dạng giá trị nhập vào một giá trị giữa 0 và 100 và trả về một mẫu tự theo thang tỉ lệ như dưới đây:*

5.16 *Sử dụng mô hình chương trình được minh họa dưới đây. Hãy xác định phạm vi của tất cả các biến khai báo trong C*

5.17 *Phạm vi của các biến sau đây trong mô hình chương trình được minh họa dưới đây là bao nhiêu. Hãy tìm các biến tổng thể có thể được tham chiếu trong cả phần chính và cả hàm 1?*

5.18 *Trong ngôn ngữ lập trình của họ C nếu Value 1 là một biến nguyên thì nghĩa của dấu &value1 là gì?*

5.19 *Hãy viết các câu lệnh C phù hợp tương ứng với các tình huống sau đây:*

(a) iptr là một con trỏ đến một biến nguyên

(b) biến được trỏ đến bởi iptr là một số nguyên.

(c) Biến được trỏ đến bởi cptr là một biến ký tự

(d) Biến số nguyên mà địa chỉ của nó nằm trong giá trị _ptr.

5.20 *Khảo sát tập hợp vị trí bộ nhớ sau đây và các lệnh kiểu C. Nội dung của các biến và các con trỏ sau khi thực thi các lệnh được minh họa dưới đây theo tuần tự nào? Giả sử tất cả các biến có kiểu giống nhau và rằng các biến Ptr là những con trỏ trỏ đến kiểu các biến đó.*

5.21 *Hãy viết một thủ tục con khi C++ để thực thi bài tập 5.15, viết thủ tục con với hai tham số, tham số đầu tiên là một tham số nhập vào số cỡ font là 100. Tham số thứ hai được dùng để trả về cấp mẫu tự tương ứng. Hãy truyền tham số thứ nhất theo giá trị và trường tham số thứ hai theo tham chiếu.*

5.22 *Hãy viết một thủ tục con C++ để chấp nhận dưới dạng nhập vào một số nickels, dimes và quarters và trả về lượng dolla tương ứng. Truyền biến tương ứng với lượng đô la bằng cách tham chiếu.*

5.23 *Hãy viết một hàm Java để trả về một giá trị gấp đôi của số dặm trên một gallon ứng với một khoảng cách nguyên đã cho. Các gallon nguyên được dùng làm một tham số.*

5.24 *Hãy viết một phương pháp Java thực thi bài tập trong bài tập có lời giải 5.22. Hàm này sẽ chấp nhận một số các nickels, dimes và quarters và trả về lượng đô la tương ứng. Hãy nhớ rằng Java chỉ có thể trả về một giá trị trực tiếp bằng cách sử dụng hàm trả về.*

SUPPLEMENTARY PROBLEMS
Các bài tập bổ sung

5.25 The following narrative describes the task of a subprogram. Can this task be accomplished by a single function? What tasks seem to be more appropriate for a subroutine than for a function?

"The subprogram reads a sequence of characters and verifies that there are no duplicate words next to each other unless they are separated by a period. If there are duplicate words the program deletes the second word. In addition, the program should verify that the first word following a period begins with a capital letter."

5.26 The following narrative describes the task of a subprogram. Can this task be accomplished by a single function?

"The subprogram receives as its input a pointer to a character string and returns the number of words in the string. A word is any sequence of alphanumeric characters separated by at least one blank. Some of the words are written in uppercase letters."

5.27 Example 5.15 swaps the values of two integer variables with the parameters being passed by reference. Can this subprocedure swap the two input parameters if the parameters are passed by value instead?

5.28 Write the prototype for the following C functions:

(a) Function name: Cubic Root
Return data type: double
Input data types: two input values of type double

(b) Function name: Copy String
Return data type: pointer to a character
Input data type: pointer to character string, pointer to character string, integer value

(c) Function name: Polynomial
Return data type: none
Input data type: pointer to character string, pointer to character string, pointer to character string, pointer to character string

5.29 What is the difference between formal and actual parameters?

5.30 Write function headers for the following Visual Basic functions:

(a) Function name: ConvertToCharacter
Return data type: character
Input data type: an integer value between 0 and 9

(b) Function name: TimeDifference

Return data type: string of characters

Input data type: a couple of strings of the form hh:min:secs

5.31 If variables ptrl and ptr2 are both integer pointer variables, what is the difference, if any, between the statements ptr1 = ptr2 and *ptr1 = *ptr2?

5.32 Is it true that for any variable x, the expression *&x can be considered synonymous?

5.33 Write a Java method that implements the problem in Solved Problem 5.13, a function that receives two integer values as parameters and determines whether the first value is a multiple of the second. Java supports the simple data type "boolean."

5.34 Write a Java function to implement the grade problem in Solved Problem 5.15.

5.35 What does the Mystery function do?

```
#include<stdio.h>
void Mystery (int, int *);
void main ()
{
   int a,b;
   a=12; b=15;
   Mystery (a,&b);
   printf("\n%d %d\n",a,b);
}
void Mystery(int x, int* y)
{
   int z;
   z=x;
   *y=*y+z;
   x=*Y;
}
```


5.36 Mark the order of execution of the statement in the following program:

```
#include <stdio.h>
void f1 (int *, int *) ;
void main()
{
   int val1,val2;
```

```
    printf("\n Write the order of execution"); // No._____
    f1(&val1,&val2); // No._____
    printf("\nThe value read in are %4d and %4d", val1,val2); // No._____
}
void f1 (int * i, int * j)
{
    printf("\nEnter the first integer number:"); // No._____
    scanf("%d",i); // No._____

    printf("\nEnter the second integer number:"); // No._____
    scanf("%d",j); // No._____

    *i=*i+2; // No._____
    *j=*j+5; // No._____
}
```

5.37 Write a C++ function that takes as parameters three integers, and returns the minimum of the three.

5.38 Write a C function to calculate the power i^n where i and n are both integers.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

BÀI TẬP BỔ SUNG

5.25 *Phần trình bày dưới đây mô tả tác vụ của một chương trình con. Tác vụ này có thể được hoàn thành bởi một hàm đơn không? Các tác vụ này dường như phù hợp đối với một thương trình con thay vì một hàm phải vậy không?*

5.26 *Phần sau đây mô tả tác vụ của một chương trình con. Tác vụ này có thể được hoàn thành bởi một hàm hay không?*

5.27 *Ví dụ 5.15 cho ra các giá trị của hai biến nguyên với các tham số được tải theo tham chiếu. Chương trình con này có thể trao đổi hai tham số nhập vào nếu các tham số được truyền theo giá trị thay thế được không?*

5.28 *Hãy viết nguyên mẫu dành cho các hàm C sau đây:*

5.29 *Có sự khác biệt nào giữa các tham số hình thức và tham số thực tế?*

5.30 *Hãy viết các tiêu đề hàm (header hàm) của các hàm Visual Basic sau đây:*

5.31 *Nếu các biến ptr1 và ptr2 cả hai đều là các biến con trỏ nguyên thì có sự khác biệt nào giữa các câu lệnh ptr1=ptr2 và *ptr1=*ptr2 hay không?*

5.32 *Câu phát biểu dưới đây có đúng hay không ứng với bất kỳ biến x nào thì biểu thức *&x được xem như là đồng nghĩa?*

5.33 *Hãy viết một phương pháp Java để thực thi bài tập trong bài toán có lời giải 5.13. Một hàm nhận hai giá trị nguyên làm các tham số và xác định xem thử giá trị đầu tiên có phải là bội số của giá trị thứ hai hay không? Java hỗ trợ kiểu dữ liệu đơn giản "boolean".*

5.34 *Hãy viết một hàm số Java để thực thi bài toán về cấp độ trong bài tập có lời giản 5.15.*

5.35 *Hàm Mystery thực hiện chức năng gì?*

5.36 *Hãy đánh dấu thứ tự thực thi của câu lệnh trong chương trình sau đây:*

5.37 *Hãy viết một hàm C++ nhận làm tham số ba số nguyên và trả về giá trị cực tiểu trong ba số đó.*

5.38 *Hãy viết một hàm C để tính i^n, trong đó i và n cả hai đều là số nguyên.*

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

5.25 No! A function should perform single tasks. This narrative requires that multiple tasks be performed by a single function. There should be at least two subroutines: one for reading the sequence of characters and one for deleting consecutive double words. A function can be used to determine if two consecutive words are identical. However, a subprocedure is more appropriate for this task since the subprocedure may "return" a Yes/No indication that the words are duplicate and, if they are, a pointer to the word that needs to be eliminated.

5.26 Yes! This subprogram performs a single task. The function should return the number of words in the given string.

5.27 No! When the parameters are passed by value the subprocedure operates on copies of the input parameters. Therefore, any changes to the formal parameters will not be reflected on the actual parameters.

5.28 *(a)* double CubicRoot (double, double);

(b) char *CopyString (char *, char *, int);

(c) void Polynomial (char *, char *, char *, char *);

5.29 Formal parameters are used in the declaration of the function or subprocedure. They are local to the subprogram in which they are defined. Actual parameters are the values passed to the subprogram.

5.30 *(a)* Private Function ConvertToCharacter (iValue As Integer) As String

(b) Private Function TimeDifference (stTunel As String, stTime2 As String) As String

5.31 These two sets of statements are totally different. Assume that the content of variable ptrl is 10FF The statement ptr1 = ptr2 makes both variables point to the same location. That is, the content of both variables is 10FF The statement *ptr1 = *ptr2 copies the value of the memory location pointed to by ptr2 into the location pointed to by ptr1. That is, assume that the content of the variable pointed to by ptr2 is 555. After this instruction is executed the variable pointed to by ptrl is also 555.

5.32 True. The expression *&x always refers to the content of variable x.

5.33
```
public boolean IsMultiple(int x, int y)
{
if (x%y==0) return true;
else return false;
}
```

5.34
```
public char GetLetterGrade(int grade) //receives integer as parameter, returns char
{
char lettergrade='',
 if (grade>=90 && grade<=100)
       lettergrade='A';
 else if (grade<90 && grade>=80)
       lettergrade='B';
else if (grade<80 && grade>=70)
       lettergrade='C';
else if (grade>=60 && grade<70)
       lettergrade='D';
else if (grade<60 && grade>=0)
lettergrade='F';
return lettergrade;
```

535 The function Mystery adds the contents of variables a and b and assigns the sum to variable b.

5.36
```
void main()
{
   int val1,val2;
   printf("\n Write the order of execution"); // No. 1
   f1(&val1,&val2); // No. 2
   printf("\nThe value read in are %4d and %4d",
   val1,val2); // No. 9
}
void f1 (int * i, int * j)
{
   printf("\nEnter the first integer number:"); // No. 3
   scanf("%d",i); // No.4
   printf("\nEnter the second integer number:"); // No. 5
```

```
    scanf("Ad",j); // No. 6
*i=*i+2; // No. 7
*j=*j+5; // No. 8
```

5.37

```
int min_of_three_integers (int a, int b, int c)
{ int minimum;
   minimum=a;
   if (b<minimum)
      minimum=b;
   else
   if ( c<minimum)
      minimum=c;
return (minimum); }
```

5.38

```
int Power (int num, int exp)
{
   int i;
   int product=1;
   for (i=1; i<=exp; i++)
      product=product*num;
   return product;
}
```

CHAPTER 6

Arrays and Strings

Mảng và chuỗi

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các khái niệm về mảng và chuỗi, các phương pháp xử lý mảng và chuỗi trong các ngôn ngữ lập trình C/C++, Visual Basic và Java, với các nội dung cụ thể sau đây:

- Introduction to arrays
- Arrays in Visual Basic
- Arrays in C/C++ and Java
- Searching
- Sorting

- *Giới thiệu sơ lược về mảng*
- *Các mảng trong Visual basic*
- *Các mảng trong C/C++ và Java*
- *Tìm kiếm*
- *Sắp xếp thứ tự*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.

CHỦ ĐIỂM 6.1

INTRODUCTION TO ARRAYS
Giới thiệu sơ lược về mảng

An **array** is a group of memory locations all of the same type that have the same name. Previously, in a program to calculate employee pay, the user would type in the employee number, the pay rate, and the number of hours worked. There would be one variable to hold each piece of information as shown in Fig. 6-1, and then the gross pay would be calculated.

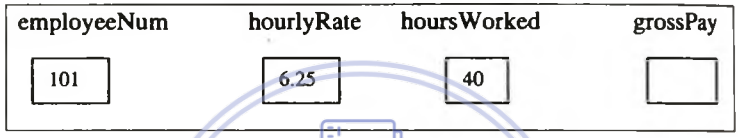


Fig. 6-1 Individual variables for payroll.

If there were more than one employee, the program could have a loop for the user to type in the information into the same variables for the second employee and calculate that person's gross pay, then the third, and so forth. However, a problem would arise if the employer wanted to have a report containing a list of the weekly pay for all the employees, the average pay for the week, and then a list of all the employees who received above the average pay. The average could not be calculated until all the employees' information was submitted. In order to compare each person's pay to the average, all the information would need to be entered a second time.

Oane solution to this problem would be to have a different variable for each employee, as shown in Fig. 6-2. Each person's pay could be compared to the average. This would be possible if there were only two or three employees, but completely impractical to declare separate variables for twenty or a hundred or a thousand employees.

The solution to this problem is to use an array. Arrays allow the storing and manipulating of large amounts of data. Three arrays are declared, one to hold all the employee numbers, another for all the hourly rates and a third for all the hours worked, as shown in Fig. 6-3. The information is entered in such a way that the employee whose number is in box 2 receives the rate in box 2 and worked the number of hours in box 2. Each person's gross pay is calculated and the average is found. Each person's data is still in memory and can quickly be examined to produce the list of people with pay above the average.

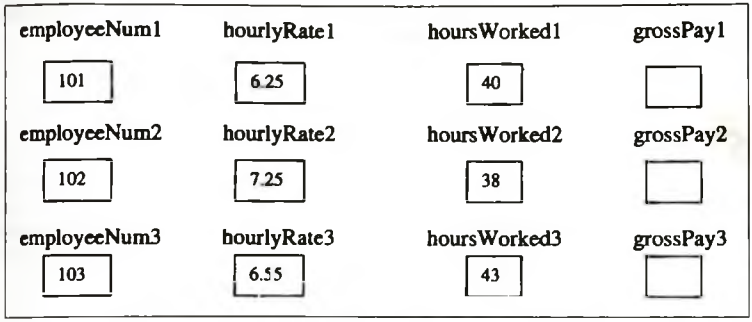


Fig. 6-2 Individual variables for three employees.

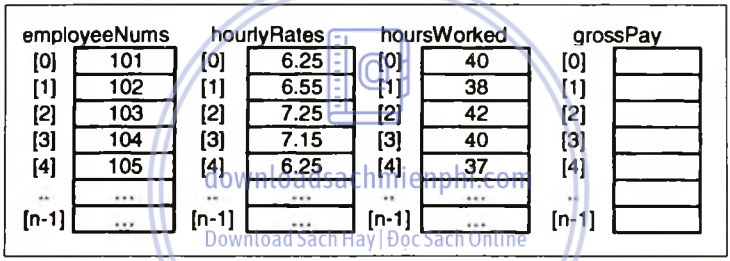


Fig. 6-3 Array variables for any number (n) of employees.

6.1.1 Manipulating Arrays - *Xử lý các mảng*

The entire array is declared once and known by one name. When it is declared, the exact number of memory locations to be set aside is specified. All the locations must contain data of the same type. Each ***element***, or individual memory location in the array, is accessible through the use of its ***subscript*** or index number. For example, in Fig. 6-3 employeeNums[3] refers to the element in box number 3 of the employeeNums array which contains "104," or hourlyRates[4] would access the element in box 4 of the hourlyRates array which contains "6.25." The subscripts usually begin with 0.

Input and output for an array is accomplished through the use of loops. Each time through the loop a different box in the array is filled or printed.

EXAMPLE 6.1 Pseudocode for a loop to **fill** or print an array of n items would look like this:

```
loop from lcv = 0 to lcv = n -1 where n is the total number of items in the array
    input or output array[lcv]
end loop
```

EXAMPLE 6.2 Most other processing of arrays also entails loops. One common example is to calculate the sum of all the items in an integer array. Pseudocode for summing an array would look like this:

```
set the sum to 0 before the loop starts
loop from lcv = 0 to lcv = n -1 where n is the total number of items in the array
    add the array[lcv] to the running sum
end loop
```

Specific processing of arrays in Visual Basic, C, C++, and Java is demonstrated later in this chapter. Each language handles them in a slightly different way. However, in all languages the programmer must be careful not to try to process past the end of the array. For example, if there are 10 items in the array called grossPay, located in boxes [0] through [9], a statement including grossPay[20] would cause serious problems because there is no memory location with that designation.

6.12 Multi-Dimension Arrays - Các mảng đa chiều

Regular single-dimension arrays are good for processing lists of items. Sometimes, however, two-dimensional arrays are necessary.

mySales

	[0]	[1]	[2]
[0]	250	300	325
[1]	350	325	400
[2]	220	315	210
[3]	210	310	295

Fig. 6-4 Two-dimensional array for sales.

EXAMPLE 6.3 A sales representative might have a report of the sales for each month of each quarter, as shown in Fig. 6-4. The rows [0] through [3] represent the four quarters of the year. The columns [0] through [2] represent the three months in each quarter. Each element of the array is

accessed through two subscripts, one for the row and one for the column, in that order. For example, in Fig. 6-4 the contents of entry mySales[1][2] is "400."

Two-dimensional arrays follow the rules of single-dimension arrays. The array is declared specifying the number of memory locations desired by stating the number of rows and the number of columns. Each item in the array must contain the same type of data. The entire array has one name and each element is accessible through the use of its row and column numbers.

Most processing of these arrays is accomplished through the use of nested loops, one for the row and one for the column.

EXAMPLE 6.4 The pseudocode for printing a two-dimensional array looks like this:

```
loop from row = 0 to row = n -1 where n is the number of rows in the array
    loop from col = 0 to col = m -1 where m is the number of columns in the array
        print out array[row][col]
    end col loop
end row loop
```

Arrays -of more than two dimensions are possible, but rarely used. See Solved Problems 6.4 and 6.5 at the end of the chapter for an example.

6.1.3 Strings - A Special Kind of Array - Chuỗi - *Một kiểu mảng đặc biệt*

The array examples we have seen so far have been numeric. It is also possible to manipulate arrays of characters. These arrays, usually called strings, are a special type of array because they are used so frequently. An array of strings is implemented as a two-dimensional array of characters.

EXAMPLE 6.5 Draw single-dimension arrays to contain the following strings: "JOAN," "CHICAGO," and "MAY" In addition, draw a two-dimensional array to contain the strings: "corn," "wheat," and "rye."

The result is shown in Fig. 6-5.

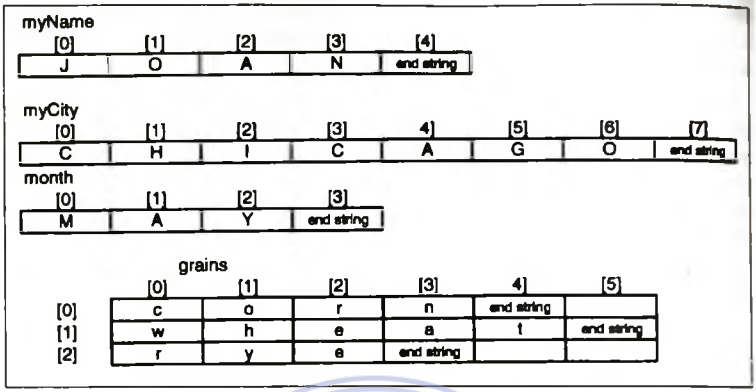


Fig. 6-5 Strings in one and two dimensions.

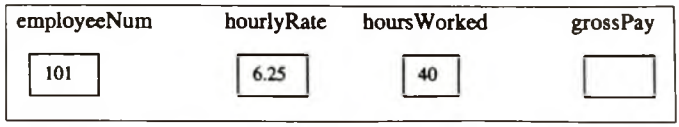
Each language implements and manipulates strings differently. Usually special kinds of processing commands are available. Many require some indication of where the string ends, as shown in the previous example.

The following sections explain the use of arrays in specific languages. In each of these languages we have considered the following topics: declaring arrays, manipulating arrays, two-dimensional arrays, string processing, and arrays as parameters to functions.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 6.1

6.1. GIỚI THIỆU VỀ MẢNG

Một mảng là một nhóm các vị trí trong bộ nhớ có cùng kiểu và cùng tên. Trước đây trog một chương trình để tính số tiền phải chi trả cho nhân viên người dùng cần phải gõ nhập mã số của nhân viên, số tiền chi trả và số giờ làm việc. Sẽ có một biến để giữ mỗi mảng thông tin như minh họa trong hình 6.1, rồi sau đó số tiền chi trả gộp được tính toán.

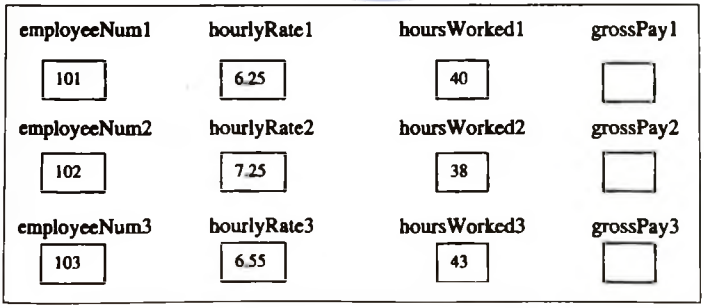


Hình 6.1 *Các biến riêng biệt dành cho chi trả lương*

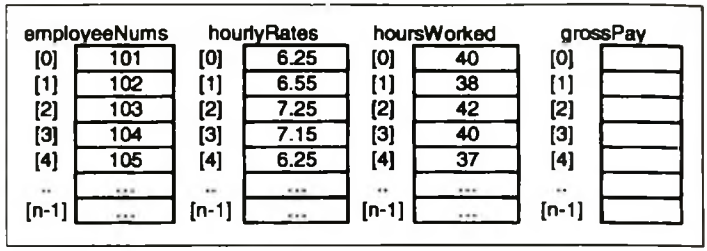
Nếu có nhiều nhân viên thì chương trình có thể có một vòng lặp dành cho người dùng để gõ nhập thông tin vào các biến giống nhau dành cho người công nhân thứ hai rồi tính khoảng tiền gộp mà người đó nhận được, sau đó đến người thứ ba v,v.... Tuy nhiên, một bài toán nảy sinh là nếu người chủ muốn có một bảng báo cáo có chứa danh sách chi trả hàng tuần đối với tất cả các nhân viên, mức chi trả trung bình mỗi tuần mộtdanh sách tất cả các nhân viên nhận số tiền bên trên mức chi trả trung bình. Số tiền trung bình không thể được tính toán đến khi tất cả các nhân viên được nhập xong. Để so sánh mức nhận tiền của mỗi người với mức trung bình, tất cả thông tin cần phải được nhập vào lần thứ hai.

Một giải pháp cho vấn đề này đó là bạn phải có một biến khác dành cho mỗi nhân viên như minh họa trong hình 6.2. Mỗi khoảng chi trả cho nhân viên phải được so sánh với giá trị trung bình. Điều này sẽ có thể thực hiện được nếu chỉ có hai hoặc ba nhân viên, nhưng nó hoàn toàn không thực tế khi bạn khai báo các biến riêng biệt dành cho khoảng 20 hoặc 100 hoặc 1000 nhân viên.

Có một giải pháp cho vấn đề này đó là sử dụng một mảng. Các mảng cho phép lưu trữ cách xử lý một lượng dữ liệu lớn. Ba mảng được khai báo, một để giữ tất cả các số nhân viên, một dùng cho tấtcả các định mức chi trả theo giờ và thứ ba là dành cho tất cả giờ làm việc như minh họa trong hình 6.3. Thông tin được nhập vào theo một cách thức sao cho nhân viên có một con số định danh nằm trong ô 2 thì nhận được định mức lương trong ô 2. và hoạt động số giờ nằm trong ô 2. Mỗi khoản tiền chi trả gộp của cá nhânsẽ được tính toán và tìm giá trị trung bình. Mỗi dữ liệu cá nhân vẫn nằm trong bộ nhớ và có thể nhanh chóng được xem xét tạo nên một danh sách những người có mức chi trả bên trên mức trung bình.



Hình 6.2 *Các biến riêng biệt dành cho ba nhân viên*



Hình 6.3 *Biến mảng dành cho bất cứ số nhân viên nào.*

6.1.1 Xử lý các mảng

Mảng tổng thể sẽ được khai báo một lần và được biết dưới một tên. Lúc được khai báo, số chính xác của các vị trí bộ nhớ sẽ được xác lập ở nơi được chỉ định. Tất cả vị trí phải có chứa dữ liệu cùng kiểu. Mỗi yếu tố (element) hoặc vị trí bộ nhớ riêng biệt trong mảng phải được thông qua việc sử dụng chỉ số hoặc số subscript. Ví dụ, trong hình 6.3, employeeNums[3] ám chỉ đến những yếu tố trong ô số 3 của mảng employeeNUms có chứa "104", hoặc hourlyRates[4] sẽ truy cập yếu tố trong ô 4 với mảng hourlyRates có chứa "6.25". Các chỉ số thường bắt đầu bằng số 0.

Dữ liệu nhập vào và xuấtra dành cho một mảng được hoàn thành thông qua việc sử dụng các vòng lặp. Mỗi lần thông qua vòng lặp có một ô khác nhau trong mảng được điền vào hoặc được in.

VÍ DỤ 6.1 *Tạo mã giả cho một vòng lặp để điền hoặc in một mảng n hạng mục như dưới đây.*

```
loop from lcv = 0 to lcv = n -1 where n is the total number of items in the array
    input or output array[lcv]
end loop
```

VÍ DỤ 6.2 *Hầu hết việc xử lý các mảng cũng chi tiết hóa các vòng lặp. Một ví dụ phổ biến đó là tính tổng của tất cả các hạng mục trong một mảng nguyên. Lập mã giả để tính tổng một mảng như dưới đây.*

```
set the sum to 0 before the loop starts
loop from lcv = 0 to lcv = n -1 where n is the total number of items in the array
    add the array[lcv] to the running sum
end loop
```

Quy trình xử lý chuyên biệt của các mảng trong Visual Basic, C++ và Java được minh họa về sau trong chương này. Mỗi ngôn ngữ sẽ xử lý chúng theo một cách thức hơi khác nhau. Tuy nhiên, trong tất cả các ngôn ngữ người lập trình phải cẩn thận không được xử lý vượt quá số cuối của mảng. Ví dụ, nếu có 10 hạng mục trong một mảng được gọi là grossPay, được đặt trong các ô từ [0] đến [9], một câu lệnh grossPay[20] sẽ xảy ra các vấn đề bởi vì chúng không có trong vị trí nhớ với sự thiết kế.

6.1.2 Các mảng nhiều chiều (nhiều thứ nguyên)

Các mảng một chiều bình thường thì tốt cho việc xử lý danh sách các hạng mục. Tuy nhiên, đôi khi vẫn cần đến các mảng hai chiều.

mySales

	[0]	[1]	[2]
[0]	250	300	325
[1]	350	325	400
[2]	220	315	210
[3]	210	310	295

***Hình 6.4** Mảng hai chiều cho việc kinh doanh*

***VÍ DỤ 6.3** Một người đại diện bán hàng có thể có một bản báo cáo về việc kinh doanh hàng tháng của mỗi quý, như minh họa trong hình 6.4. Các hàng từ [0] đến [3] đại diện cho 4 quý trong năm . các cột từ [0] đến [12] đại diện cho 3 tháng trong mỗi quý. Mỗi thành phần của mảng được tiếp cận thông qua 2 chỉ số, một chỉ số cho hàng và một chỉ số cho cột theo thứ tự. Ví dụ, ở hình 6.4 , nội dung của mySales[1][2] nhập vào là 400.*

Các mảng hai chiều tuân theo quy tắc của các mảng một chiều. Mảng này được khai báo bằng các xác định số vị trí nhớ mong muốn khi khai báo số hàng hay số cột. Toàn bộ mảng có một tên và mỗi thành phần có thể tiếp cận được bằng cách dùng số hàng và số cột của nó.

Phần lớn quy trình xử lý mảng được thực hiện bằng cách sử dụng các vòng lặp lồng, một vòng lặp cho hàng và một vòng lặp cho cột.

***VÍ DỤ 6.4** Mã giả để in một mảng hai chiều có dạng như sau:*

```
loop from row = 0 to row = n -1 where n is the number of rows in the array
    loop from col = 0 to col = m -1 where m is the number of columns in the array
        print out array[row][col]
    end col loop
end row loop
```

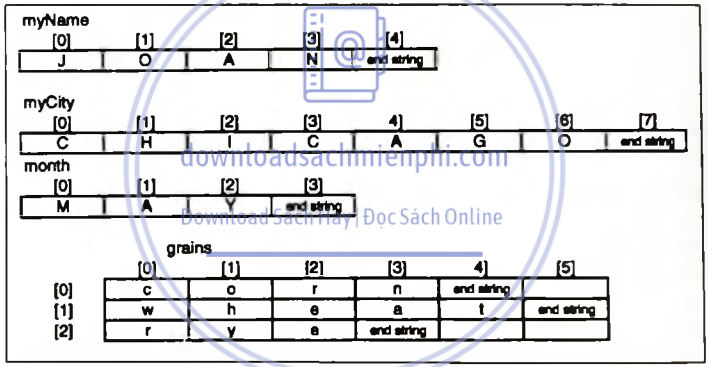
Cũng có thể có khả năng nhiều hơn hai chiều nhưng rất ít được sử dụng để xem ví dụ.. Xem phần các bài tập có lời giải 6.4 và 6.5 ở cuối chương

6.1.3 Các chuỗi - Một loại mảng đặc biệt

Các ví dụ về mảng mà chúng ta đã xem xét là những con số. Ngoài ra ta vẫn có thể xử lý các mảng ký tự. Những mảng này thường được gọi là các chuỗi, đây là một loại mảng đặc biệt bởi vì chúng được dùng thường xuyên. Một mảng các chuỗi được thực thi dưới dạng một mảng các ký tự hai chiều.

VÍ DỤ 6.5 *Hãy vẽ các mảng mộtchiều có chứa các chuỗi sau đây: JOAN", "CHICAGO", và "MAY". Bên cạnh đó, hãy vẽ một mảng hai chiều có chứa các chuỗi "corn", wheat", và "rye".*

Kết quả được minh họa trong hình 6.5



Hình 6.5 *Các chuỗi một chiều và hai chiều*

Mỗi ngôn ngữ thực thi và xử lý các chuỗi theo một cách thức khác nhau. Thông thường các loại lệnh xử lý đặc biệt đều có sẵn. Có nhiều loại yêu cầu một vài chỉ định nơi mà chuỗi phải kết thúc như minh họa ở ví dụ trước đây.

Phần sau đây giải thích cách dùng các mảng theo các ngôn ngữ đặc biệt. Trong mỗi một ngôn ngữ này chúng ta phải xem xét các chủ điểm sau đây: khai báo mảng, xử lý mảng, các mảng hai chiều, xử lý chuỗi và các mảng được chọn làm tham số cho các hàm..

CHỦ ĐIỂM 6.2

ARRAYS IN VISUAL BASIC
Các mảng trong Visual basic

6.2.1 Declaring Arrays - *Khai báo các mảng*

In Visual Basic, arrays are declared like other variables, using the Dim statement. The number of memory locations to be used is placed in parentheses immediately following the array name.

EXAMPLE 6.6 An integer and a floating point array would be declared in this way:

```
Dim testScores (10) As Integer 'an array to keep 10 integer
   test scores from 0 to 9

Dim cashAvailable (12) As Currency 'an array to keep 12
   months record of cash from 0 to 11
```

Unlike most other languages, Visual Basic allows the lowest subscript of the array to be set to a value other than 0. The subscripts may even be negative, but it is best to have all subscripts be integer values. For these non-zero-based arrays, both the lowest and the highest subscripts are specified in the Dim statement.

EXAMPLE 6.7 Declare an array from 1 to 12 to hold monthly income. Declare an array of integer values with subscripts from -20 to 20.

```
Dim monthlyIncome (1 to 12) As Currency     '12 items in
   the array

Dim values (-20 to 20) As Integer      '41 items in the ar-
   ray
```

If the program attempts to access an invalid element of the array, Visual Basic stops and gives an error message, Subscript out of range. If the program uses non-standard subscript ranges, it is very important to verify that the element accessed really exists.

6.2.2 Manipulating Arrays - *Xử lý các mảng*

Most Visual Basic processing with arrays is accomplished using the For ...Next loops.

EXAMPLE 6.8 Write a Visual Basic program to read in a set of ten test scores, find the average, and print out the average and the scores that are above average.

Figure 6-6 shows a section of code that would accomplish this task. In each section, the For ...Next loop goes through the entire array to fill or process each element.

```
Dim lcv As Integer    'loop control variable
Dim sum As Integer
Dim scores(1 To 10) As Integer
Dim avg As Single

'read in scores
For lcv=1 To 10       'read in the entire array
   scores(lcv)=InputBox("Enter the number")
Next lcv

'find the average
sum=0
For lcv=1 To 10
   sum=sum+scores(lcv)            'add each to sum
Next lcv
avg=sum/10

'print those above average
Print "The average is"; avg
For lcv=1 To 10
  If scores(lcv)>avg Then         'only print the ones above average
    Print scores(lcv)
  End If
Next lcv
```


Fig. 6-6 Visual Basic test scores.

6.2.3 Two-dimensional Arrays - *Các mảng hai chiều*

Two-dimensional arrays are declared with the row boundaries and the column boundaries in the same parentheses, separated by a comma. In each case, if no lower boundary is specified, a lower bound of zero is assumed.

EXAMPLE 6.9 Declare an array to keep track of daily temperatures for 31 days in 12 months. Declare an array to monitor 5 test scores for each person in the class numbered from 101 to 110. Declare a two-dimensional array of strings with 4 rows and 26 columns.

```
Dim dailyTemperatures (1 to 12, 1 to 31) As Integer      '12
   monthly rows, 31 daily columns

Dim classScores (101 to 110, 1 to 5) As Integer          'stu-
   dent numbers 101 to 110, 5 scores

Dim nameTable (3, 25) As String   'rows 0 to 3, 0 to 25
```

Processing of two-dimensional arrays is accomplished using nested For..Next loops, as shown in Fig. 6-7.

EXAMPLE 6.10 Write a Visual Basic program where four weeks of temperatures are entered and printed in a well-documented chart. Remember that the Dim statement always indicates rows first and then columns. The code to implement this program is shown in Fig. 6-7.

```
Dim row As Integer, col As Integer
Dim temps (1 To 4, 1 To 7) As Single
Dim message As String
'read in temps
For row=1 To 4
  For col=1 To 7
     message="Enter the temperature for week
     "+Str(row)+" and day "+Str(col)
     temps(row, col)=InputBox(message) 'see chapter 3 to
     review explanation of message
  Next col
Next row
'print chart - heading first
Print 'DAY:", 'Sun', "Mon",    "Tue", "Wed", "Thu",
  "Fri", 'Sat'
For row=1 To 4          'do everything for each row in this
  loop
  Print "Week"; row,      'label for each row followed by
                          comma for next column
  For col=1 To 7          'do     everything for each
                          column in this loop
  Print temps(row, col), 'each temp followed by comma for
                          next column
  Next col
  Print          'take cursor to next line for next row
Next row
```

Fig. 6-7 Temperatures in Visual Basic.

The output for the code in Fig. 6-7 with sample data entered looks like this:

DAY:	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Week 1	55	56	57	59	60	52	49
Week 2	45	42	45	42	35	30	26
Week 3	21	25	26	32	36	39	40
Week 4	45	48	52	53	57	51	49

6.2.4 String Processing - *Xử lý chuỗi*

Visual Basic provides a special data type called String to handle arrays of characters. The String type makes it easier for the programmer to process because the String variable can expand or contract to be the length needed for any given String value. The programmer does not have to keep track of the end of the string. Visual Basic also provides the necessary functions to manipulate strings.

EXAMPLE 6.11 Look at this section of code.

```
Dim myName As String
myName="Joe"
Print myName; " is "; Len (myName); " characters long"
myName="Alexander the Great"
Print myName; " is ; Len (myName); " characters long"
```

The output would be:

```
Joe is 3 characters long
Alexander the Great is 19 characters long
```

The built-in Len() function returns the exact length of the string, not including the end of string mark. Visual Basic takes care of marking the end of string, making programming less complex. Arrays of strings can be declared to make the two-dimensional array of characters more understandable. Remember, in VB only the number of strings in the list need to be declared, not the length of each string. Other string processing functions available in VB include:

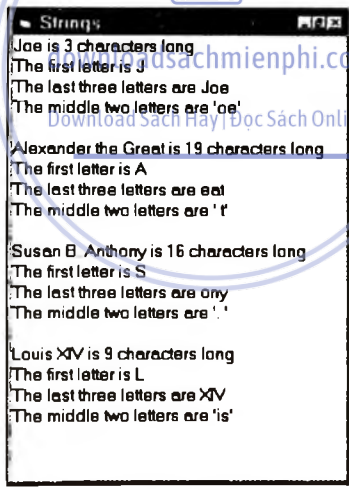
- Len(*string*) returns the length of the string.
- Right(*string*, n) returns the rightmost n characters.
- Left(*string*, n) returns the leftmost n characters.
- Mid(*string*, p, n) returns the middle n characters beginning at position p.

EXAMPLE 6.12 Write a Visual Basic program that examines an array of strings and determines the following: (a) the length of each string, (b) the leftmost character, (c) the rightmost three characters, and (d) the middle two letters. Fgure 6-8 shows the code using the Visual Basic functions.

```
Dim myNames(1 To 4) As String
Dim lcv As Integer, middle As Integer
myNames(1)="Joe"
myNames(2)="Alexander the Great"
myNames(3)="Susan B. Anthony"
myNames(4)="Louis XIV"
For lcv=1 To 4
 Print myNames(lcv); " is "; Len(myNames(lcv)); " characters long"
 Print "The first letter is "; Left(myNames(lcv), 1)
 Print "The last three letters are "; Right(myNames(lcv), 3)
 middle=(Len(myNames(lcv)) / 2)
 Print "The middle two letters are '"; Mid(myNames(lcv), middle, 2); "'"
 Print
Next lcv
```

Fig. 6-8 String processing in Visual Basic.

The output for this code is shown below. Notice that a space is considered a character. It is listed as one of the middle two characters for both "Susan B. Anthony" and "Alexander the Great."



6.2.5 Arrays as Parameters to Functions - *Các mảng có chức năng là các tham số*

An entire array can be sent to a function as a parameter in Visual Basic. This is often done if the same function needs to be applied to several different arrays to keep from repeating code.

EXAMPLE 6.13 Write a generic Visual Basic function to find the average of the numbers in an integer array. That function can receive an array of temperatures, an array of scores, or any other integer array, as long as the number of elements in the array is also sent.

Figure 6-9 shows this function and also some sample code calling that function with different arrays as parameters.

The function is:

```
Private Function FindAvg(arr() As Integer, length As Integer) As Single
 Dim sum As Integer, lcv As Integer
  For lcv=1 To length
    sum=sum+arr(lcv)
  Next lcv
  FindAvg=sum/length 'returns value of function as single
End Function
```

One part of the calling code:

```
Dim lcv As Integer 'loop control variable
Dim scores(1 To 10) As Integer
Dim numScores As Integer, numIn As Integer
Dim avg As Single
'handle scores
lcv=1
numIn=InputBox('Enter the score (-1 to stop)')
Do While (lcv<10 And numIn>0) 'stop at 10 or -1
   numScores=numScores+1
   scores(lcv)=numIn
   numIn=InputBox('Enter the score (-1 to stop)')
   lcv=lcv+1
Loop
avg=FindAvg(scores, numScores) 'send exact number of scores
Print 'The average score is '; avg
```

Another part of the calling code:

```
'handle temps
 Dim temps(1 To 7) As Integer
 For lcv=1 To 7
   numIn=InputBox('Day'+ Str(lcv)+' Enter the temperature')
   temps(lcv)=numIn
Next lcv
avg=FindAvg(temps, 7) 'send 7 days
Print 'The average temperature this week is '; avg
```


Fig. 6-9 Functions with array parameters in Visual Basic.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 6.2

6.2 CÁC MẢNG TRONG VISUAL BASIC

6.2.1 Khai báo các mảng

Trong Visual Basic các mảng được khai báo giống như các biến khác, bằng cách sử dụng câu lệnh Dim. Số các vị trí bộ nhớ được dùng và được đặt trong các dấu ngoặc đơn theo sau tên của mảng.

***VÍ DỤ 6.6** Số nguyên và một mảng dấu chấm động cần được khai báo theo cách sau đây:*

```
Dim testScores (10) As Integer 'an array to keep 10 integer
    test scores from 0 to 9

Dim cashAvailable (12) As Currency 'an array to keep 12
    months record of cash from 0 to 11
```

Không giống như tất cả các ngôn ngữ khác, Visual Basic cho phép các chỉ số dưới thấp nhất của mảng và được xác lập sang một giá trị khác 0. Các chỉ số dưới này thậm chí có thể là số âm, nhưng tốt nhất là phải có tất cả các chỉ số là giá trị nguyên đối với các mảng trên nền khác zero, cả các chỉ số thấp nhất và cao nhất đều được chỉ định trong câu lệnh Dim.

***VÍ DỤ 6.7** Khai báo một mảng từ 1 cho đến 12 để theo dõi thu nhập hàng tháng. Khai báo một mảng các giá trị nguyên với chỉ số dưới từ - 2- đến 20.*


```
Dim monthlyIncome (1 to 12) As Currency    '12 items in the array

Dim values (-20 to 20) As Integer '41 items in the array
```

Nếu chương trình thử truy cập vào một phần tử bị cấm trong mảng, thì Visual Basic ngưng và cho ta một thông báo lỗi. Subscript out of range. Nếu chương trình này sử dụng các miền subscript không chuẩn thì điều quan trọng đó là phải kiểm tra rằng phần tử được truy cập hiện đang thật sự có.

6.2.2 Xử lý các mảng

Hầu hết quy trình xử lý Visual Basic với các mảng được hoàn thành bằng cách sử đụng các vòng lặp For ... Next.

***VÍ DỤ 6.8** Hãy viết một chương trình Visual Basic để đọc trong một tập hợp 10 điểm thi, tìm điểm trung bình và in ra điểm trung bình và các điểm nằm bên trên điểm trung bình.*

Hình 6.6 trình bày một mục mã hoàn tất tác vụ này. Trong mỗi mục, vòng lặp For...Next thông qua toàn bộ mảng để lấp đầy hoặc xử lý mỗi phần tử.

```
Dim lcv As Integer    'loop control variable
Dim sum As Integer
Dim scores(1 To 10) As Integer
Dim avg As Single
'read in scores
For lcv=1 To 10       'read in the entire array
   scores(lcv)=InputBox('Enter the number')
Next lcv
'find the average
sum=0
For lcv=1 To 10
   sum=sum+scores(lcv)            'add each to sum
Next lcv
avg=sum/10
'print those above average
Print 'The average is'; avg
For lcv=1 To 10
  If scores(lcv)>avg Then         'only print the ones above average
    Print scores(lcv)
  End If
Next lcv
```


Hình 6.6 *Visual Basic kiểm tra điểm thi*

6.2.3 Các mảng hai chiều

Các mảng hai chiều được khai báo với các đường biên hàng và các đường biên cột trong cùng dấu móc đơn, tách nhau bởi dấu phẩy. Trong mỗi trường hợp. Nếu không có biên dưới được chỉ định, thì biên dưới giả sử bằng 0.

VÍ DỤ 6.9 *Hãy khai báo một mảng để theo dõi nhiệt độ hàng ngày trong 31 ngày trong vòng 12 tháng. Khai báo một mảng để giám sát 5 điểm thi cho mỗi một người trong lớp được đánh số từ 101 đến 110. Khai báo một mảng hai chiều của các chuỗi có 4 hàng và 26 cột.*

```
Dim dailyTemperatures (1 to 12, 1 to 31) As Integer      '12
   monthly rows, 31 daily columns

Dim classScores (101 to 110, 1 to 5) As Integer          'stu-
   dent numbers 101 to 110, 5 scores

Dim nameTable (3, 25) As String   'rows 0 to 3, 0 to 25
```

Việc xử lý các mảng hai chiều được hoàn thành bằng cách sử dụng các vòng lặp lồng nhóm For ... Net như minh họa hình 6.7.

VÍ DỤ 6.10 *Hãy viết một chương trình Visual Basic ở đó có 4 tuần, nhiệt độ được nhập vào và được in trong một biểu đồ. Hãy nhớ rằng*

câu lệnh Dim luôn luôn chỉ định trước tiên là các hàng và sau đó là các cột.

Mã để thực thi chương trình này được minh họa trong hình 6.7.

```
Dim row As Integer, col As Integer
Dim temps(1 To 4, 1 To 7) As Single
Dim message As String
'read in temps
For row=1 To 4
  For col=1 To 7
    message="Enter the temperature for week "+Str(row)+" and day "+Str(col)
    temps(row, col)=InputBox(message) 'see chapter 3 to review explanation of message
  Next col
Next row
'print chart - heading first
Print "DAY:", "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
For row=1 To 4              'do everything for each row in this loop
  Print "Week"; row,        'label for each row followed by comma for next column
  For col=1 To 7            'do everything for each column in this loop
    Print temps(row, col),  'each temp followed by comma for next column
  Next col
  Print                     'take cursor to next line for next row
Next row
```

***Hình 6.7** Nhiệt độ trong Visual Basic.*

Kết quả xuất của mã trong hình 6.7 với dữ liệu mẫu được nhập vào giống như dưới đây.

Temperatures

DAY:	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Week 1	55	56	57	59	60	52	49
Week 2	45	42	45	42	35	30	26
Week 3	21	25	26	32	36	39	40
Week 4	45	48	52	53	57	51	49

6.2.4 Xử lý chuỗi

Visual Basic cung cấp một kiểu dữ liệu đặc biệt được gọi là String để xử lý các mảng ký tự. Kiểu String giúp cho các nhà lập trình dễ dàng xử lý bởi vì biến String có thể mở rộng hay thu hẹp chiều dài cần thiết cho bất cứ giá trị String đã cho nào. Người lập trình không cần phải theo dõi phần cuối của chuỗi. Visual Basic cũng cung cấp các hàm cần thiết để xử lý chuỗi.

***VÍ DỤ 6.11** Hãy xem một mã dưới đây.*

```
Dim myName As String
myName="Joe"
Print myName; " is "; Len (myName); " characters long"
myName="Alexander the Great"
Print myName; " is ; Len (myName); " characters long"
```

Kết quả xuất sẽ là

```
Joe is 3 characters long
Alexander the Great is 19 characters long
```

Hàm Len() đượctạo sẵn để trả về chiều dài chính xác của chuỗi, nó không chứa phần cuối điểm đánh dấu chuỗi. Visual Basic xem xét thận trọng việc đánh dấu phần cuối của chuỗi, điều này làm cho việc lập trình ít phức tạp.

Các mảng chuỗi có thể được khai báo để tạo nên mảng ký tự hai chiều dễ hiểu hơn. Hãy nhớ rằng, chỉ trong VB thì số các chuỗi trong danh sách mới cần được khai báo, chứ không phải chiều dài của chuỗi.

Các hàm xử lý chuỗi khác có sẵn trong VB bao gồm:

- *Len(string) trả về chiều dài của chuỗi.*
- *Right(string, n) trả về n ký tự bên phải nhất*
- *Left(string, n) trả về ký tự bên trái nhất*
- *Mid(string, n) trả về n ký tự ở giữa bắt đầu tại vị trí p.*

***VÍ DỤ 6.12** Hãy viết một chương trình Visual Basic để xem xét một mảng các chuỗi và xác định nội dung sau đây: (a) chiều dài của mỗi chuỗi, (b) ký tự bên trái nhất, (c) ba ký tự bên phải nhất và (d) hai ký tự ở giữa.*

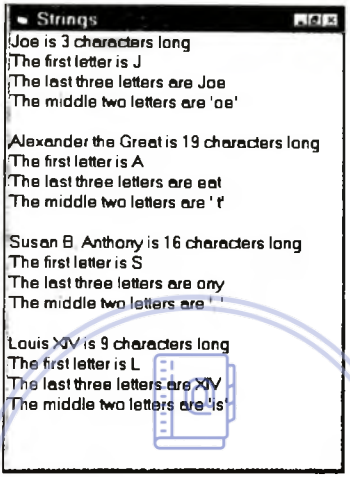
Hình 6.8 minh họa mã bằng cách sử dụng các hàm Visual Basic.

```
Dim myNames(1 To 4) As String
Dim lcv As Integer, middle As Integer
myNames(1)="Joe"
myNames(2)="Alexander the Great"
myNames(3)="Susan B. Anthony"
myNames(4)="Louis XIV"
For lcv=1 To 4
 Print myNames(lcv); " is "; Len(myNames(lcv)); " characters long"
 Print "The first letter is "; Left(myNames(lcv), 1)
 Print "The last three letters are "; Right(myNames(lcv), 3)
 middle=(Len(myNames(lcv)) / 2)
 Print "The middle two letters are " "; Mid(myNames(lcv), middle, 2); """
 Print
Next lcv
```

***Hình 6.8** Xử ý chuỗi trong Visual Basic*

Đầu ra của mã này được minh họa dưới đây. Lưu ý rằng một khoảng trống được xem là một ký tự. Nó được liệt kê dưới dạng một trong số

hai ký tự ở giữa dùng cho cả Susan B. Anthony" và "Alexander the Great".



6.2.5 Các mảng được chọn làm tham số cho các hàm

Một mảng tổng thể có thể được gởi sang một hàm dưới dạng một tham số trong Visual Basic. Điều này thường được thực hiện nếu cùng một hàm giống nhau thì được áp dụng cho nhiều mảng khác nhau để giữ mã khỏi lặp lại.

***VÍ DỤ 6.13** Hãy viết một hàm Visual Basic tổng quát để tìm giá trị trung bình của các số trong một mảng số nguyên. Hàm đó có thể nhận một mảng nhiệt độ, một mảng điểm số hoặc bất cứ mảng số nguyên nào khác (miễn là số phần tử trong mảng được gửi đến.*

Hình 6.9 biểu thị hàm này và một mã mẫu gọi đến hàm đo với các mảng khác nhau làm tham số.

The function is:

```
Private Function FindAvg(arr() As Integer, length As Integer) As Single
 Dim sum As Integer, lcv As Integer
  For lcv=1 To length
    sum=sum+arr(lcv)
  Next lcv
  FindAvg=sum/length 'returns value of function as single
End Function
```

One part of the calling code:

```
Dim lcv As Integer 'loop control variable
Dim scores(1 To 10) As Integer
Dim numScores As Integer, numIn As Integer
Dim avg As Single

'handle scores
lcv=1
numIn=InputBox("Enter the score (-1 to stop)")
Do While (lcv<10 And numIn>0) 'stop at 10 or -1
   numScores=numScores+1
   scores(lcv)=numIn
   numIn=InputBox("Enter the score (-1 to stop)")
   lcv=lcv+1
Loop
avg=FindAvg(scores, numScores) 'send exact number of scores
Print "The average score is "; avg
```

Another part of the calling code:

```
'handle temps
 Dim temps(1 To 7) As Integer
 For lcv=1 To 7
   numIn=InputBox("Day"+ Str(lcv)+" Enter the temperature")
   temps(lcv)=numIn
Next lcv
avg=FindAvg(temps, 7) 'send 7 days
Print "The average temperature this week is "; avg
```


Hình 6.9 Các hàm với các tham số mảng trong Visual Basic

CHỦ ĐIỂM 6.3

ARRAYS IN C/C++ AND JAVA
Các mảng trong C/C++ và Java

Recall from Chapter 5 that pointers are variables that contain memory addresses as their values. A variable name *directly* references a value and a pointer *indirectly* references the value.

6.3.1 Declaring Arrays in C/C++ - *Khai báo mảng trong C/C++*

A pointer to an array is shown in Fig. 6-10. When implementing arrays in C, C++, and Java, the array name is the same as a pointer that points to the FIRST object in the array.

Figure 6-10 demonstrates how an array can be declared in C and C++. The first statement:

```
int ar[6];
```

tells the compiler to set aside 6 memory locations (0 through 5) for integer data types. All subscripts in C and C++ begin with 0. Each location is specified through the use of square brackets (e.g., array[2]) instead of the parentheses used in VB.

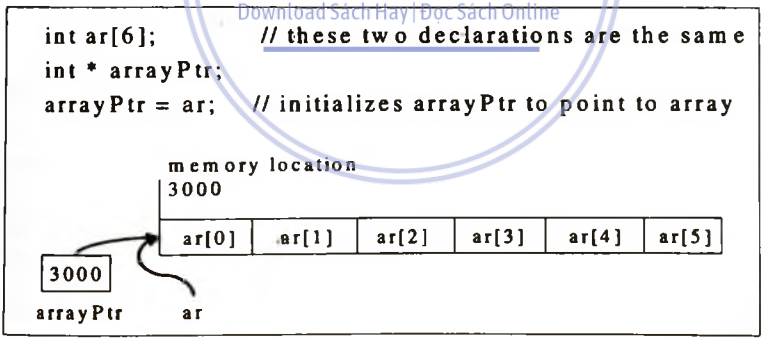


Fig. 6-10 Pointers and array names in C and C++.

EXAMPLE 6.14 Write a statement that would set up the array of 6 spaces and actually put values into each space:

```
int ar[6]={5,10,15,20,25,30};
```

It is possible to declare and initialize C and C++ arrays at the same time. The compiler usually gives an error if more than the specified num-

ber of values are listed. If fewer than that number are listed, many compilers will fill in the rest with zero.

It is also possible to initialize all elements of the array to zero at the same time, like this:

```
int ar[6]={0};
```

6.3.2 Dedaring Arrays in Java - *Khai báo mảng trong Java*

Declaring an array in C/C++ allocates the appropriate number of memory locations. Declaring arrays in Java is more explicit. First, the program declares the array and then it allocates memory. This task can be accomplished either in -one step or in two.

EXAMPLE 8.15 Write the declaration from Fig. 6-10 in Java.

```
int ar [ ] ;      // declaring array
  ar=new int[6];      // allocating memory
or int ar[]=new int[6]; // declaring and allocating in one statement
```

It is possible to declare, allocate, and initialize. Java arrays at the same time. This statement would set up the array of 6 spaces and actually put values into each space:

```
int ar[]={2, 3, 4, 1, 2, 6};
int ar[]={0};
```


The compiler only allocates the number of items that are listed. The first array above has 6 items, 0 through 5. The second has only 1, array space zero. Java never automatically allocates or fills any memory locations. That is the responsibility of the program.

63.3 Manipulating Arrays in C, C++, and Java - *Xử lý mảng trong C, C++ và Java*

As in Visual Basic, arrays are usually ,processed in C, C++, and Java using loops. One of the dangerous aspects of C++ is that if the loop attempts to access locations not in the array, most compilers will not give an error message. The program simply uses whatever happens to be in that location, which is often undefined. In computer lingo, we refer to the content of these locations as "garbage." Programmers must be very careful, especially when using loops to access the array items, to be sure that the loop does not go beyond the end of the array.

EXAMPLE 6.16 Write a short section of code in C++ that declares an array of six integers with the value of each element double its index and then try to print ten elements of the array.

Figure 6-11 illustrates this code.

```
int ar[6], i;
for (i=0; i<6; i++)
   ar[i]=2*i;
for (i=0; i<10; i++)
   cout<<ar[i]<<" ";
cout<<endl<<endl;
```

Fig. 6-11 Manipulating C++ arrays.

Notice that the array was declared with 6 elements and subscripts 0 through 5. However, the second loop tries to print out up to the subscript 9. Beyond the sixth location, the contents are garbage. Some C and C++ compilers will simply print out in integer form whatever happens to be in those locations. The output might look like this:

0 2 4 6 8 10 6684216 4208633 1 7867264

Once the array was accessed past where it had been initialized, whatever happened to be in those particular locations was printed out. If the array is used in calculations of any kind the results would be completely unpredictable. Java offers more help to the programmer. Trying to access any locations beyond the bounds of an array always results in an error message. The message below specifies that the program tried to go beyond the array, and it was the index 5 that caused the problem.

Java.Lang.ArrayIndexOutOfBoundsException: 5

In order to prevent this kind of error in C and C++, many programmers use constants to signify the number of locations, and then use the constant as a boundary in every loop. Using this convention in Java also results in code that is much clearer.

EXAMPLE 6.17 Write a short section of code to declare an array of 6 integers using the constant MAX and use this constant to control execution of a loop that initializes the contents of each element to the same value as its index.

The C/C++ and Java code would look like this:

C/C++

```
const int MAX=6;
int ar[MAX], i;
for (i=0; i<MAX; i++)
  ar[i]=i;
```

Java

```
final int MAX=6;
int ar[], i;
ar=new int[MAX];
for (i=0; i<MAX; i++)
  ar[i]=i;
```

EXAMPLE 6.18 Write a C++ program that reads in integers and keeps track of how many are entered.

Another way to keep track of the items in the array is to allocate a separate variable containing the number of items in the array. Figure 6-12 demonstrates this using the variable size to keep track of how many are entered into the array. The constant MAX assures that the program never processes past the end of the array. The same concept could be implemented in Java.

```
const int MAX=6;
int ar[MAX], i;
int num, size=0;
cout<<"Enter a number, -1 to stop"<<endl;
cin>>num;
while (num !=-1 && size<MAX) //stop the loop with -1 or if it gets too big
{
   ar[size]=num;
   size++;
   cout<<"Enter a number, -1 to stop"<<endl;
   cin>>num;
}
for (i=0; i<size; i++)
   cout<<ar[i]<<" ";
cout<<endl<<endl;
```


Fig. 6-12 Keeping track of the size of an array in C++.

6.3.4 Two-dimensional Arrays - *Các mảng hai chiều*

Two-dimensional arrays in C, C++, and Java require the row and column size in separate brackets.

EXAMPLE 6.19 Write declarations in C/C++ and Java for an array with 3 rows and columns, and an array with 4 rows and 6 columns.

C/C++	Java
`int mySquare[3][3];`	`int mySquare[ ][ ];`
	`mySquare=new int [3][3];`
`int myTable[4][6];`	`int myTable[ ][ ];`
	`myTable=new int [4][6];`

EXAMPLE 6.20 Write the statements in C/C++ and Java that declare two two-dimensional arrays and initialize them as they are declared. Use the following data: (a) the contents of each element in each row of the first array is the row number plus 1, (b) the contents of each element of the second array is the (row number times 10) plus (the column number plus 1).

Two-dimensional arrays can also be initialized as they are declared in C, C++, and Java. Figure 6-13 shows an initialization in C/C++ and one in Java, as well as the resulting arrays.

C/C++

```
int mySquare[3][3]={ {1,1,1},{2,2,2}, {3,3,3} };
```

Java

```
int myTable[][]={ {1,2,3,4,5,6}, {11,12,13,14,15,16},
      {21,22,23,24,25,26}, {31,32,33,34,35,36} };
```

	[0]	[1]	[2]
[0]	1	1	1
[1]	2	2	2
[2]	3	3	3

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	1	2	3	4	5	6
[1]	11	12	13	14	15	16
[2]	21	22	23	24	25	26
[3]	31	32	33	34	35	36

Fig. 6-13 Initializing two-dimensional arrays.

EXAMPLE 6.21 Given the following arrays, write the nested loops that will initialize them as indicated below.

	[0]	[1]	[2]
[0]	1	2	3
[1]	2	4	6
[2]	3	6	9

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	1	2	3	4	5
[1]	1	2	3	4	5	6
[2]	2	3	4	5	6	7
[3]	3	4	5	6	7	8

As in Visual Basic, processing is done using nested loops, one for the row and one for the column. The loops shown in Fig. 6-14, which would be the same in C, C++, and Java, show these two initializations using nested loops.

```
for (row=0; row<3; row++)
  for (col=0; col<3; col++)
    mySquare[row][col]=(row +1)*(col+1);
for (row=0; row<4; row++)
    for (col=0; col<6; col++)
       myTable[row][col]=row+col;
```

Fig. 6-14 Processing two-dimensional arrays.

6.3.5 String Processing in C and C++ - *Xử lý chuỗi trong C và C++*

The processing of strings is the one area completely different in C/C++ and Java, so they will be addressed in separate sections. In C and C++, string processing is very cumbersome. There is no separate data type called string. The **string** is an array of characters, ending in the null character '\0'. An array of strings is a two-dimensional array of characters, with each row ending in the null character '\0'. Recall that some C/C++ compilers allow array elements to be processed that are beyond the bounds of the declared array. When dealing with strings, this can be very frustrating. C/C++ provides a string-handling library that sometimes helps in handling strings, but the programmer still needs to be very careful that the null character actually is present to indicate the end of each string.

EXAMPLE 6.22 The short C++ program in Fig. 6-15 demonstrates some of the C/C++ string-handling capabilities. Each program section is explained below.


```
#include<iostream.h>
#include <string.h>
void main ()
{
    //Section 1
    char word[3];
    strcpy (word,'my');    //alternate declaration:    char * word='my';
    cout<<word<<endl;
    cout<<strlen(word)<<endl;    //prints out 2
    //Section 2
    char word2[]={'S','t','o','r','y','\0'};
    cout<<word2<<endl;
    cout<<strlen(word2)<<endl; //prints out 5
    //Section 3
    char word3[]='is good';
    cout<<word3<<endl;
    cout<<strlen(word3)<<endl;    //prints out 7
    if (strcmp(word2, word3)) cout<<'They are not the same'<<endl;
    else cout<<'The strings are identical'<<endl;
    //Section 4
    char *greetings[3]={'I love you', 'Be mine', 'Valentine'};
    for (int i=0; i<3; i++)
        cout<<greetings[i]<<' is '<< strlen(greetings[i])<< ' characters long'<<endl;
    //the strings are 10, 7, and 9 characters long respectively
    cout<<endl;
}
```

Fig. 6-15 String example in C++.

The first section shows that the most common way of declaring a string is as a simple array of characters. Be sure that the array size is one more than the number of characters in the word to allow for the null character at the end. In this example, word is declared as 3 characters long which does include the null character. Once the array is declared, it cannot be filled by using a simple assignment statement. A loop could be used, but the string.h library provides for the string copy strcpy(destination, source) function. The strcpy() function automatically appends the null character at the end of the string. The example also shows an alternate type of declaration. An array of characters can also be declared without specifying its length through the use of the asterisk to indicate a pointer to a group of characters. Many programmers find it easier not to be required to specify the length of the string. However, it is usually easier and safer in C/C++ to declare the length explicitly. An array of characters is the one kind of array that can be printed out without accessing each item in the array individually. The cout statement knows to print the characters until it comes to the null character. The strlen(string) function returns the number of characters in the string, NOT COUNTING the null character. If the programmer wants to process each character in the string by using a loop, the strlen could be used as the upper bound as in this code:

```
for (i=0; i<strlen(word); i++) cout<<word[i];
```

The second section demonstrates how to declare a string and initialize it at the same time. If each character is listed with single quotes, then the null character MUST be listed explicitly at the end. Note that C/C++ is consistent in enclosing a character in single quotation marks and a string in double quotation marks.

The third section shows an alternate way of declaring a string and initializing it at the same time. If the string is listed as a single group of characters surrounded by double quotation marks, the null character is automatically added at the end. Note that a space is a character and is counted in the length of the string.

The strcmp (strl, str2) is a string-handling function that compares two strings to see if they are the same or different. The function returns a zero if they are the same, a 1 if the first string is greater than the second one, and a -1 if the second string is greater than the first one. The ASCII value of each character (see Chapter 3) is compared. Therefore, all capital letters are less than all the lowercase letters. Therefore, the programmer must be sure that the case is the same for each string being compared.

The last section demonstrates an array of strings. In reality, this statement is declaring an array of 3 pointers, each pointing to a character at the beginning of a string. The actual two-dimensional array is shown in Fig. 6-16. The brackets indicate the array, and the asterisk indicates that the array is of pointers to characters. Each element in the array of strings can be printed individually using cout.

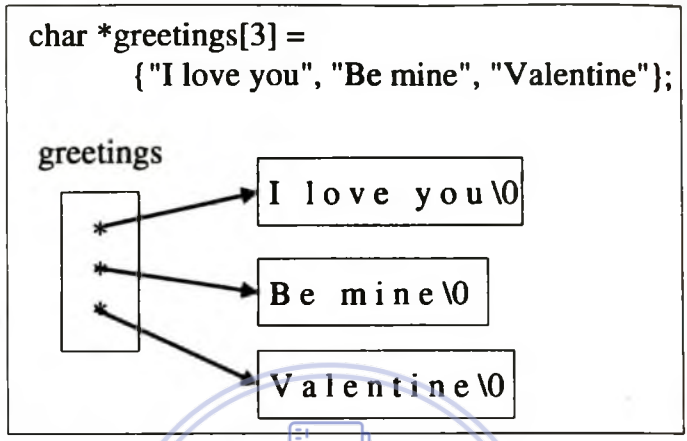


Fig. 6-16 Array of strings in C/C++.

6.3.6 String Processing in Java - *Xử lý chuỗi trong Java*

String processing in Java is much different from C/C++. There is a built-in String class (see Chapter 8 for a complete explanation of classes) which behaves much like the String data type in Visual Basic. However, unlike C/C++, the programmer does not need to deal with the null character indicating the end of the string. Because this class is quite complex, only the most basic string processing will be explained in this section.

EXAMPLE 6.23 Figure 6-17 shows some Java code implementing strings. Each program section will be explained below.

```
//Section 1
String myStr="me";                  //declare and allocate memory
System.out.println (myStr);
System.out.println (myStr.length());
//Section 2
String newStr;                      //declare
newStr=new String("you");           //allocate memory
System.out.println (newStr);
System.out.println (newStr.length());
if (myStr.equals(newStr)) System.out.println("The strings are identical");
else System.out.println("They are not the same");
//Section 3
String listOfNames[]=new String[3];    //declare array of Strings
listOfNames[0]=("Robert Redford");     //allocate memory
listOfNames[1]=("Willie Nelson");
listOfNames[2]=("Sophia Loren");
for (int i=0; i<3; i++)
  System.out.println(listOfNames[i]+" is "+ listOfNames[i].length()+" characters long");
```

Fig. 6-17 String example in Java

The first section demonstrates how to declare a String, allocate memory and initialize it in one statement. The String can report its own length through the use of the *StringName.length()* method (Java's term for functions belonging to a class).

The next section declares a String in one line and then allocates memory and initializes the String in the next line. There are a number of String comparing methods available to each string. The one demonstrated here, *Stringl.equals(String2), is* a Boolean method that compares the two strings using the ASCII chart to see if they are equal. It returns *true* if the strings are equal and *false* if they are not. Another method available, *Stringl.equalsIgnoreCase(String2), is* often of more value than the pure equals().

The last section declares an array of Strings, and then allocates memory and initializes each element. Notice that each item in the array can use the methods available to any String. Many other methods are available to the String class. See Fig. 6-30 for another example.

6.3.7 Arrays as Parameters to Functions – *Các mảng thực hiện chức năng là tham số*

An entire array can be sent to a function as a parameter in C, C++, and Java, just as in Visual Basic. Remember that the name of the array is just a pointer, or reference, to the first item in the array. Therefore, arrays are always sent by reference. Any changes made to the array in the function will be retained in the calling code.

EXAMPLE 6.24 Write a generic function to find the average of the numbers in an integer array. That function can receive an array of temperatures, an array of scores, or any other integer array, as long as the number of elements in the array is also sent.

Figure 6-18 illustrates this function.

The Java, C, and C++ function is the same, and the Java version of the calling program listed shows only minor differences from the C++ version. Notice that only the array name is included in the calling statement. The function heading contains the brackets to indicate that an array is an incoming parameter. Other parts of the code are self-explanatory.

```
//C, C++ and Java functions are identical
float FindAvg(int arr[], int length)
{
   int sum=0, lcv;
   for(lcv=0; lcv<length; lcv++)
      sum=sum+arr[lcv];
   return (float)sum/length;     //returns value of function as float
}
//portion of C++ calling program
   const int MAX=6;
   int ar[MAX], i;
   for (i=0; i<MAX; i++)
      ar[i]=i+10;
   float avg=FindAvg(ar, MAX);
//portion of Java calling program
   final int MAX=6;              //difference in declaration of constant
   int ar[]=new int[MAX];        //difference in declaration of array
   int i;
   for (i=0; i<MAX; i++)
      ar[i]=i+10;
   float avg=FindAvg(ar,MAX);
```

Fig. 6-18 Arrays as parameters to functions.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 6.3

6.3 CÁC MẢNG TRONG C/C++ VÀ JAVA

Hãy nhớ lại trong chương 5 rằng các con trỏ là các biến có chứa các địa chỉ như là địa chỉ của chúng. Một tên biến trực tiếp tham chiếu đến một giá trị và một con trỏ gián tiếp tham chiếu đến giá trị.

6.3.1 Khai báo mảng trong C/C++

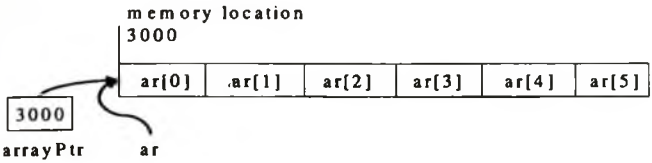
Một con trỏ trỏ đến một mảng được minh họa trong hình 6.10. Khi thực thi mảng trong C, C++, và Java, tên mảng cũng là một con trỏ trỏ đến đối tượng đầu tiên trong mảng.

Hình 6.10 minh họa cách thức mà một mảng được khai báo trong C và C++. Câu lệnh đầu tiên:

int ar[6];

báo cho trình biên soạn đặt 6 vị trí nhớ (từ 0 đến 5) với kiểu dữ liệu là số nguyên. Tất cả các chỉ số trong C và C++ bắt đầu bằng 0. Mỗi vị trí được xác định thông qua việc sử dụng các dấu móc (ví dụ array[2] thay cho dấu ngoặc đơn được dùng trong VB.)

```
int ar[6];          // these two declarations are the same
int * arrayPtr;
arrayPtr = ar;      // initializes arrayPtr to point to array
```



Hình 6.10 *Các pointer và các tên mảng trong C và C++*

VÍ DỤ 6.14 *Hãy viết một câu lệnh nhằm xác lập mảng 6 khoảng trống và đặt các giá trị thực tế vào mỗi khoảng trống:*

```
int ar[6] = {5, 10, 15, 20, 25, 30};
```

Ta có thể khai báo và khởi tạo các mảng C và C++ tại cùng một thời điểm. Trình biên soạn thường báo một lỗi nếu có nhiều số giá trị được liệt kê. Còn nếu số giá trị ít hơn con số được liệt kê thì trình biên soạn sẽ lấp đầy vào phần còn lại với giá trị zero.

Chúng ta có thể khởi tạo tất cả các yếu tố của mảng sang zero tại cùng một thời điểm như dưới đây.

```
int ar [6] ={0};
```

6.3.2 Khai báo các mảng trong Java

Việc khai báo một mảng trong C/C++ cấp phátsố cac1 vị trí bộ nhớ phù hợp. Việc khai báo các mảng trong Java thì rõ ràng hơn. Trước tiên, chương trình khai báo mã sau đó cấp phát bộ nhớ. Tác vụ này có thể được hoàn thành hoặc trong một hoặc hai bước.

VÍ DỤ 6.5 *Hãy viết phần khai báo từ hình 6.10 trong java.*

```
int ar[];                // declaring array
ar=new int[6];           // allocating memory
```

hoặc

```
int ar[]=new int[6];     // declaring and allocating in one statement
```

Ta có thể khai báo, cấp phát, và khởi tạo các mảng Java tại cùng một lúc. Câu lệnh này sẽ xác lập mảng 6 khoảng trống và thật sự đặt các giá trị vào trong mỗi mảng.

```
int ar[]={2, 3, 4, 1, 2, 6};
int ar[]={0};
```

Trình biên soạn chỉ cấp phát số các hạng mục được liệt kê. Mảng đầu tiên trên đây có 6 hạng mục, từ 0 cho đến 5. Mảng thứ hai chỉ có 1, khoảng trống mảng là zero. Java không bao giờ tự động cấp phát hoặc lấp đầy bất cứ vị trí bộ nhớ nào. Đây chính là trách nhiệm của chương trình.

6.3.3 Xử lý các mảng trong C, C++ và Java

Cũng như trong Visual Basic, các mảng thường được xử lý trong C, C++, và Java bằng cách sử dụng các vòng lặp. Một trong những hía cạnh nguy hiểm của C++ đó là nếu vòng lặp thử truy cập các vị trí không nằm trong mảng thì hầu hết các trình biên soạn không đưa ra một thông điệp báo lỗi. Chương trình đơn giản sử dụng bất cứ những gì xảy ra trong vị trí đó, thường là không được xác định. Nói theo ngôn ngữ của máy tính, chúng ta tham chiếu đến nội dung của những vị trí này dưới dạng các (vật thừa thải). Những nah 2lập trình cũng phả rất thận trọng đặc biệt lúc sử dụng các vòng lặp để truy cập các hạng mục mảng, cần bảo đảm rằng vòng lặp không vượt xa khỏi phần cuối của mảng.

VÍ DỤ 6.16 *Hãy viết một đoạn mã ngắn trong C++ để khai báo một mảng có 6 số nguyên với giá trị của mỗi một phần tử gấp đôi chỉ số của nó rồi thử in mười phần tử của mảng. Hình 6.11 minh họa mã này.*


```
int ar[6], i;
for (i=0; i<6; i++)
    ar[i]=2*i;
for (i=0; i<10; i++)
    cout<<ar[i]<<" ";
cout<<endl<<endl;
```

Hình 6.11 *Xử lý các mảng C++*

Lưu ý rằng mảng được khai báo với 6 phần tử và các chỉ số từ 0 đến 5. Tuy nhiên, vòng lặp thứ hai thư 3in ra lân đến chỉ số 9. Vượt xa vị tríthứ 6, các nội dungnày là dư thừa. Một vài trình biên soạn C và C++ đơn giản sẽ in ra dưới dạng số nguyên bất cứ những gì xảy ra ở những vị trí này. Kết quả xuất sẽ giống như dưới đây.

0 2 4 6 8 10 6684216 4208633 1 7867264

Một khi mảng đã được truy cập ngang qua nơi mà nó được khởi tạo, thì bất cứ điều gì xảy ra nằm trong vị trí đặc biệt này đều được in ra.

Nếu mảng được dùng trong các phép tính thuộc bất kỳ loại nào, thì các kết quả sẽ hoàn toàn ngoài dự đoán. Java đưa ra nhiều công cụ trợ giúp cho các nhà lập trình. Bằng cách thử truy cập bất cứ vị trí nào mà vượt xa biênc ủa một mảng sẽ luôn luôn cho kết quả với một thông báo lỗi. Thông điệp dưới đây chỉ chỉ định rằng chương trình thử vượt xa mảng của nó và chính chỉ số 5 gây nên sự cố.

java.lang.ArrayIndexOutOfBoundsException: 5

Để ngăn chặn kiểu lỗi này trong C và C++, nhiều nhà lập trình sử dụng các hằng số để gán cho số các vị trí rồi sử dụng hằng số là một biên trong mỗi vòng lặp. Bằng cách sử dụng quy ước này trong Java ta cũng có các kết quả đó là mã được rõ ràng hơn.

***VÍ DỤ 6.17** Hãy viết một đoạn mã ngắn để khai báo một mảng 6 số nguyên bằng cách sử dụng hằng số MAX rồi sử dụng hằng số này để điều khiển việc thực thi một vòng lặp khởi tạo các nội dung của mỗi một yếu tố mà giá trị của nó giống hệt như chỉ số.*

C/C++ và mã Java giống như dưới đây.

C/C++

```
const int MAX=6;
int ar[MAX], i;
for (i=0; i<MAX; i++)
  ar[i]=i;
```

Java

```
final int MAX=6;
int ar[], i;
ar=new int[MAX];
for (i=0; i<MAX; i++)
    ar[i]=i;
```


***VÍ DỤ 6.18** Hãy viết một chương trình C++ để đọc theo các số nguyên và theo ôi số lượng được nhập vào. Một cách khác để theo dõi các hang mục trong mảng đó là cấp phát một biến riêng biệt có chứa số các hang mục trong mảng. Hình 6.12 minh họa việc này bằng cách sử dụng kích cỡ biến để theo dõi số lượng được nhập vào mảng. Hằng số MAX bảo đảm rằng chương trình không bao giờ xử lý quá phần cuối của mảng. Khái niệm này cũng được thực thi trong Java.*

6.3.4 Các mảng hai chiều

Các mảng hai chiều trong C, C++, và Java yêu cầu kích thước hàng và cột phải được đặt trong các dấu ngoặc riêng biệt.

***VÍ DỤ 6.19** Hãy viết phần khai báo trong C/C++ và Java dành cho một mảng có 3 hàng và cột, và một mảng có 4 hàng và 6 cột.*

C/C++

```
int mySquare[3][3];

int myTable[4][6];
```

Java

```
int mySquare[ ][ ];
mySquare=new int [3][3];
int myTable[ ][ ];
myTable=new int [4][6];
```

```
const int MAX=6;
int ar[MAX], i;
int num, size=0;
cout<<"Enter a number, -1 to stop"<<endl;
cin>>num;
while (num !=-1 && size<MAX) //stop the loop with -1 or if it gets too big
{
   ar[size]=num;
   size++;
   cout<<'Enter a number, -1 to stop'<<endl;
   cin>>num;
}
for (i=0; i<size; i++)
   cout<<ar[i]<<" ";
cout<<endl<<endl;
```

Hình 6.12 *Theo dõi kích thước của một mảng trong C++*

VÍ DỤ 6.20 *Hãy viết các câu lệnh trong C và C++ và Java để khai báo 2 mảng hai chiều và khởi tạo chúng khi chúng được khai báo. Bằng cách sử dụng dữ liệu sau đây: (a) nội dung của mỗi một thanh2 phần trong mỗi hàng của mảng đầu tiên chính là số hàng cộng thêm 1, (b) các nội dung của mỗi thành phần của mảng thứ hai chính là (số hàng nhân 10) cộng thêm (số cột cộng 1).*

Các mảng hai chiều cũng có thể được khởi tạo ngay khi chúng được khai báo trong C, C++ và Java. Hình 6.13 minh họa sự khởi tạo trong C/C++ và khởi tạo trong Java, cũng như các mảng kết quả.

C/C++

```
int mySquare[3][3]={ {1,1,1},{2,2,2}, {3,3,3} };
```

Java

```
int myTable[][]={ {1,2,3,4,5,6}, {11,12,13,14,15,16},
      {21,22,23,24,25,26}, {31,32,33,34,35,36} };
```

	[0]	[1]	[2]
[0]	1	1	1
[1]	2	2	2
[2]	3	3	3

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	1	2	3	4	5	6
[1]	11	12	13	14	15	16
[2]	21	22	23	24	25	26
[3]	31	32	33	34	35	36

Hình 6.13 *Khởi tạo các mảng hai chiều.*

VÍ DỤ 6.21 *Cho các mảng sau đây, hãy viếp các vòng lồng nhóm vốn sẽ khởi tạo chúng như được chỉ định dưới đây.*

	[0]	[1]	[2]
[0]	1	2	3
[1]	2	4	6
[2]	3	6	9

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	1	2	3	4	5
[1]	1	2	3	4	5	6
[2]	2	3	4	5	6	7
[3]	3	4	5	6	7	8

Cũng như trong Visual Basic, việc xử lý được thực hiện bằng cách sử dụng các vòng lặp kết nhóm, một vòng dành cho hàng và một vòng dành cho cột.

Các vòng lặp được minh họa trong hình 6.14, nó sẽ giống hệt nhau trong C, C++ và Java, hình này biểu thị hai khởi tạo bằng cách sử dụng các vòng lặp lồng nhóm.

```
for (row=0; row<3; row++)
  for (col=0; col<3; col++)
    mySquare[row][col]=(row +1)*(col+1);
for (row=0; row<4; row++)
    for (col=0; col<6; col++)
       myTable[row][col]=row+col;
```


***Hình 6.14** Xử lý các mảng hai chiều*

6.3.5 Xử lý chuỗi trong C av2 C++

Quy trình xử lý các chuỗi là một lĩnh vực hoàn toàn khác hẳn nhau trong C/C++ và Java, vì vậy mà chúng ta sẽ khảo sát các mục riêng biệt. Trong C và C++, việc xử lý chuỗi rất đơn giản. Không có kiểu dữ liệu tách rời được gọi là chuỗi. Chuỗi chính là một mảng các ký tự, kết thúc bằng kýt ự trống '\0'. Một mảng các chuỗi là một mảng các ký tự có hai chiều, trong đó mỗi hàng kết thúc bằng ký tự trống '\0'. Hãy nhớ lại rằng một vài trình biên soạn C/ C++ cho phép các yếu tố của mảng được xử lý vượt ra ngoài các biên của mảng được khai báo. Lúc xử lý với các chuỗi, điều này rất phức tạp. C/C++ cung cấp một thư viện xử lý chuỗi để đôi lúc giúp xử lý các chuỗi, nhưng người lập trình vẫn rất thật trọng khi mà ký tự trống thật sự hiện diện để chỉ định hẳn kết thúc của mỗi chuỗi.

***VÍ DỤ 6.22** Chương trình C++ rút ngắn trong hình 6.15 minh họa một vài khả năng xử lý chuỗi C/C++ . Mỗi mục của chương trình được giải thích dưới đây.*

```
#include<iostream.h>
#include <string.h>
void main ()
{
    //Section 1
    char word[3];
    strcpy (word,'my');    //alternate declaration:    char * word='my';
    cout<<word<<endl;
    cout<<strlen(word)<<endl;    //prints out 2
    //Section 2
    char word2[]={'S','t','o','r','y','\0'};
    cout<<word2<<endl;
    cout<<strlen(word2)<<endl; //prints out 5
    //Section 3
    char word3[]='is good';
    cout<<word3<<endl;
    cout<<strlen(word3)<<endl;    //prints out 7
    if (strcmp(word2, word3)) cout<<'They are not the same'<<endl;
    else cout<<'The strings are identical'<<endl;
    //Section 4
    char *greetings[3]={'I love you', 'Be mine', 'Valentine'};
    for (int i=0; i<3; i++)
       cout<<greetings[i]<<' is '<< strlen(greetings[i])<< ' characters long'<<endl;
    //the strings are 10, 7, and 9 characters long respectively
    cout<<endl;
}
```


***Hình 6.15** Ví dụ chuỗi trong C++*

Mực đầu tiên cho thấy rằng cách phổ biến nhất để khai báo một chuỗi đó là dưới dạng một mảng các ký tự đơn giản. Cần chắc chắn rằng kích thước của mảng phải lớn hơn số các ký tự trong từ để cho phép ký tự trống nằmở cuối. Trong ví dụ này, từ được khai báo dưới dạng dài 3 ký tự không có chỗ cho ký tự trống. Một khi mảng được khai báo, nó không thể được lấp đầy bằng cách sử dụng một câu lệnh gán đơn giản. Vòng lặp có thể được dùng, nhưng thư viện string.h cung cấp cho hàm strcpy (destination, source) để sao chuỗi. Hàm strcpy() tự động đính ký tự trống vào cuối chuỗi.

Ví dụ này cũng minh họa một kiểu khai báo khác. Một mảng các ký tự cũng có thể được khai báo mà không cần chỉ định chiều dài của nó thông qua việc sử dụng ký tự thay thế (asterisk) để chỉ định một con trỏ trỏ đến một nhóm các ký tự. Nhiều nhà lập trình thấy dễ dàng hơnkhi không được yêu cầu phải chỉ định chiều dài của chuỗi. Tuy nhiên, thông thường cách dễ dàng và an toàn trong C /C++ đó là phải khai báo dễ dàng chiều dài.

Một mảng các ký tự lẽ một lợi mảng có thể được in ra mà không cần truy cập vào mỗi một mụng mục trong mảng riêng biệt. Câu lệnh cout

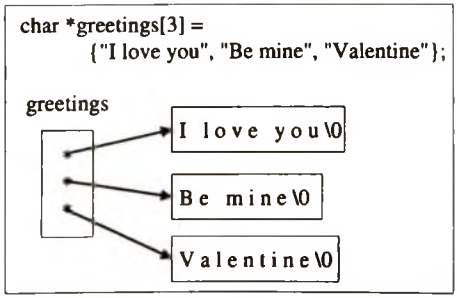
biết để in các ký tự cho đến khi nó đạt đến đến ký tự null. Hàm strlen(string0 trả về số các ký tự trong chuỗi, không tính đến ký tự trống. Nếu người lập trình muốn xử lý mỗi ký tự trong chuỗi bằngc ách sử dụng một vòng lặp thì strlen có thể được dùng làm biên trên như trong mã dưới đây.

```
for (i = 0; i<strlen(word); i +=) count<<word[i];
```

Phần thứ hai minh họa cách để khai báo một chuỗi và khởi tạo nó cùng lúc. Nếu mỗi ký tự được liệt kê trong dấu móc đơn, thì ký tự tự trống phải được liệt kê một cách rõ ràng ở cuối. Lưu ý rằng C/C++ nhất quán với nhau trong việc đặt một ký tự trong các dấu móc đơn và một chuỗi trong các dấu móc kép.

Phần thứ ba biểu thị một cách khác để khai báo một chuỗi và khởi tạo nó cùng lúc. Nếu chuỗi được liệt kê dưới dạng một nhóm các ký tự được bao quanh bởi các dấu móc kép, thì ký tự null được tự động thêm vào cuối. Lưu ý rằng một khoảng trống chính là một ký tự và nó phải được kể đến trong chiều dài của chuỗi.

strcmp(str1, str2) là một hàm xử lý chuỗi nó so sánh hai chuỗi để xem thứ chúng giống nhau hay khác nhau. Hàm này trả về một số zero. Nếu chúng giống nhau, trả về 1 nếu chuỗi đầu tiên lớn hơn chuỗi thứ hai và trả về - 1 nếu chuỗi thứ hai lớn hơn chuỗi thứ nhất. Giá trị ASCII của mỗi ký tự (xem chương 3) cũng được so sánh. Do đó, tất cả các mẫu tự hoa đều nhỏ hơn tất cả các mẫu tự thường. Vì thế người lập trình cần phải bảo đảm rằng kiểu chữ phải giống nhau trong một chuỗi đang được so sánh.Phần cuối cùng trình bày một mảng các chuỗi. Trong thực tế câu lệnh này đang khai báo một mảng 3 pointer, mỗi pointer trỏ đến một ký tự ở đầu của một chuỗi. mảng hai chiều thật sự được minh họa trong hình 6.16. Các dấu ngoặc chỉ ra mảng và dấu asterish chỉ ra rằng mảng này là các con trỏ để trỏ đến các ký tự. Mỗi yếu tố trong mảng các chuỗi có thể được in một cách riêng biệt bằng cách sử dụng count.



***Hình 6.16** Mảng các chuỗi trong C/C++*

6.3.6 Xử lý chuỗi trong Java

Xử lý chuỗi trong Java thì khác nhiều so với xử lý chuỗi trong C/C++. Có một lớp String được tạo sẵn (xem chương 8 có phần giải thích hoàn chỉnh về các lớp) nó có tính chất y hệt như kiểu dữ liệu String trong Visual Basic. Tuy nhiên, không giống như C/C++, người lập trình không cần phải xử lý ký tự null để chỉ định phần kết thúc của chuỗi. Bởi vì lớp này hoàn toàn phức tạp, cho nên chỉ có việc xử lý chuỗi căn bản nhất mới được giải thích trong mục này.

VÍ DỤ 6.23 *Hình 6.17 biểu thị một vài mã Java đang thực thi các chuỗi. Mỗi một chương trình sẽ được giải thích dưới đây.*

```
//Section 1
String myStr="me";                  //declare and allocate memory
System.out.println (myStr);
System.out.println (myStr.length());
//Section 2
String newStr;                      //declare
newStr=new String("you");           //allocate memory
System.out.println (newStr);
System.out.println (newStr.length());
if (myStr.equals(newStr)) System.out.println("The strings are identical");
else System.out.println("They are not the same");
//Section 3
String listOfNames[]=new String[3];    //declare array of Strings
listOfNames[0]="Robert Redford";       //allocate memory
listOfNames[1]="Willie Nelson";
listOfNames[2]="Sophia Loren";
for (int i=0; i<3; i++)
  System.out.println(listOfNames[i]+" is "+ listOfNames[i].length()+" characters long");
```

Hình 6.17 *Ví dụ chuỗi trong Java*

Mục đầu tiên minh hoạ cách khai báo một String, cấp phát bộ nhớ và khởi tạo nó trong một câu lệnh. String có thể báo cáo chiều dài của nó thông qua việc sử dụng phương pháp String Name.length() (thuật ngữ của Java dành cho các hàm thuộc về một lớp).

Phần kế tiếpkhai báo một String trong một dòng rồi cấp phát bộ nhớ và khởi tạo String trong vòng kế tiếp. Có một số các phương pháp String có sẵn cho mỗi chuỗi. Ở đây minh họa một phương pháp đó là , String 1.equals(String2), đây là một phương pháp Boolean để so sánh hai chuỗi bằng cách sử dụng sơ đồ ASCII xem thử chúng có bằng nhau hay không. Nếu trả về true nếu các chuỗi này bằng nhau và trả về false nếu chúng không bằng nhau. Một phương pháp khác cũng có sẵn, đó là phương pháp String1.equalsIgnoreCase (String2), nó thường cho nhiều giá trị hơn là dấu bằng thuần túy().

Phần cuối cùng khai báo một mảng các String, rồi cấp phát bộ nhớ và khởi tạo mỗi phần tử. Lưu ý rằng mỗi hạng mục trong chuỗi có thể sử dụng các phương pháp có sẵn cho bất cứ String nào. Cũng có

nhiều phương pháp có sẵn cho lớp String. Xem hình 6.30 để có ví dụ khác.

6.3.7 Các mảng làm tham số cho các hàm

Một mảng tổng thể có thể gởi đến một hàm dưới dạng một tham số trong C, C++ và Java cũng y hệt như tong Visual Basi. Hãy nhớ rằng tên của mảng chỉ là một con trỏ hoặc tham chiếu đế hạng mục dầu trong mảng. Do đổ các mảng luôn luôn được gởi bằng phương pháp tham chiếu. Bất cứ sự thay đổi nào xảy ra cho mảng trong hàm cũng sẽ được giữ lại trong mảng gọi.

***VÍ DỤ 6.24** Hãy viết một hàm tổng quát để tìm giá trị trung bình của các số trong một mảng số nguyên. Hàm đó có thể nhận một mảng nhiệt độ, một mảng các điểm hoặc bất cứ mảng nguyên nào khi số các phần tử trong mảng cũng được gởi đến. Hình 6.18 minh họa hàm này.*

***Hình 6.18** Các mảng chọn lăm tham số cho các hàm*

```
//C, C++ and Java functions are identical
float FindAvg(int arr[], int length)
{
   int sum=0, lcv;
   for(lcv=0; lcv<length; lcv++)
      sum=sum+arr[lcv];
   return (float)sum/length;          //returns value of function as float
}
//portion of C++ calling program
   const int MAX=6;
   int ar[MAX], i;
   for (i=0; i<MAX; i++)
      ar[i]=i+10;
   float avg=FindAvg(ar, MAX);
//portion of Java calling program
   final int MAX=6;                   //difference in declaration of constant
   int ar[]=new int[MAX];             //difference in declaration of array
   int i;
   for (i=0; i<MAX; i++)
      ar[i]=i+10;
   float avg=FindAvg(ar,MAX);
```

Trong Java, C và C++ hàm này là giống nhau và phi6n bản Java của chương trình gọi được liệt kê chỉ nhằm minh họa các ự khácbiệt nhỏ so với C++. Lưu ý rằng hỉ có tên mảng được đưa vào trong câu lệnh gọi. Tiêu đề hàm có chứa cácdấu ngoặc để chỉ ra rằng một mảng là một tham số gởi đến. Các phần khác của mảng tự giải thích.

CHỦ ĐIỂM 6.4

SEARCHING
Tìm kiếm

One of the main goals in programming is to save time and energy throug the reuse of code. Several generic functions like searching and sortir arrays can be written in such a way that they are useful in many differe applications. A number of different algorithms for sorting and searchir arrays have been designed and can be implemented in any language. Tv searching algorithms and three sorting algorithms will be illustrated her

6.4.1 Sequential Search - *Tìm kiếm theo trình tự*

The simplest algorithm for searching an array is the ***sequential*** searc In the sequential search the array is examined, one element at a tim until the target value is found or the end of the array is reached witho finding the target value. The sequential search is often the best choice u either of these two instances:

- The array size is small (less than 100 elements).
- The elements in the array are in no particular order.

EXAMPLE 6.25 Write a sequential search function in Visual Basic tha takes in as parameters the array to be searched, its length, and the targe value to search for. It returns the index number of the element holding th target value or -1 if the target is not in the array.

Figure 6-19 illustrates a possible implementation of this function.

```
'Sequential Search Function
 Private Function SequentialSearch(ar() As Integer, length As Integer, _
                                   target As Integer) As Integer
'This function searches 1 through length of the incoming array for the target value
'It returns the index of the element holding that value.
'If the target is not in the array, the function returns -1
  Dim lcv As Integer, targetIndex As Integer
  Dim found As Boolean
  found=False                    'set initial flag to false
  targetIndex=-1                 'default value in case target is not found
  For lcv=1 To length
    If ar(lcv)=target Then
      found=True                 'set flag to true when found
      targetIndex=lcv
    End If
 Next lcv
 SequentialSearch=targetIndex  'send back the target Index or -1
End Function
```

```
'portion of calling program
  const MAX=10
  Dim lcv As Integer          'loop control variable
  Dim response As Integer, target As Integer
  Dim scores(1 To MAX) As Integer
'read in scores
For lcv=1 To MAX
  scores(lcv)=InputBox("Enter the number")
Next lcv
'get the target value
target=InputBox("What number will you search for?")
response=SequentialSearch(scores, MAX, target)
If (response>=0) Then
   Print target; "is in the array at location"; response
Else
   Print target; "is not in the array"
End If
```

Fig. 6-19 Sequential search in Visual Basic.

There are several reasons that this algorithm is not very efficient. First, if the array had more than 100 elements, the search process would take a lot of time. Second, the loop in the search function goes through the entire array every time, even if the item is found at the first index. Third, if the target is in the array more than once, the function will return only its last index.

EXAMPLE 6.26 A small modification of the loop in the function, shown in Fig. 6-20, enables it to stop as soon as the target value is found. Instead of a For ... Next loop which executes until the lcv reaches the length, a Do While loop is used which stops as soon as the item is found. In this example, if the target is in the array more than once, the function will return only the FIRST index. An alternate search function which returns the number of times the item is in the loop is shown in Solved Problem 6.10 at the end of this chapter.

```
                                        ) function
                                        , length As Integer, _
                                        ler
                                        incoming array for the target value.
                                        t value.
                                        : returns -1


                                        to false


                                        lo while to stop when item is found




lex 'send back the target Index or -1
```

!0 More efficient sequential search.

EXAMPLE 6.27 One more modification can be made to the sequential search of the previous example to increase efficiency if the target is not in the array. If the incoming array is IN ASCENDING ORDER, as soon as an element is seen that is greater than the target, the array could stop. This requires that only one line be changed, the Do While statement:

```
Do While (lcv<=length And (Not found) AND ar(lcv) <target)
```

Figure 6-21 demonstrates the number of times the loop would execute for each of the sequential search algorithms for a target that is in the array and one that is not.

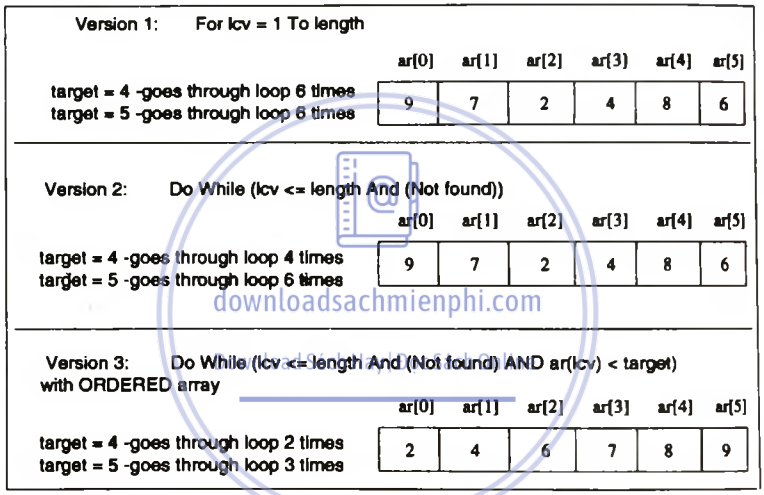


Fig. 6-21 Comparison of sequential search test conditions.

6.4.2 Binary Search - *Tìm kiếm nhị phân*

The last modification is obviously the best possible improvement on a sequential search for an ordered array. However, even that algorithm would take a long time to search a large array, such as one with 10,000 elements. A better solution for finding a target value in an ordered array with no duplicates is to use the binary search algorithm.

In the *binary search* the middle element is examined first.

- If that element is the target being sought, then the middle index is returned.
- If the middle element is less than the target, then because the array is in order the entire first half of the list can be ignored

- If the middle element is greater than the target, then the entire last half of the list is ignored.

The array sections are successively cut in half, eliminating large numbers of elements until the target is found or there is only one element in the section being examined. The binary search is usually the best choice if the array size is large. The elements in the array must be in order, so extra time would be needed if the array must be sorted first. Sorts will be examined in the next section.

EXAMPLE 6.28 Write the code necessary to implement the binary search function in C/C++. The function takes the same parameters as the sequential search: the array, the length of the array, and the target value.

Figure 6-22 illustrates this function. The Java version of this function would be identical except for the marked line.

Figure 6-23 shows a walk-through of the code for two different target values. If the array is small, no too much time is saved. If, however, the array had 1000 items, the first examination would eliminate half of the items, which would save a considerable amount of time. The binary search is a very fast search, but the array must be in order. If the array is not in order, it must be sorted before the binary search can be applied. If it cannot be sorted, then the sequential search is the only choice.


```
//C/C++ version of Binary Search (with Java change)
int BinarySearch(int ar[], int length, int target)
//This function searches 0 through length-1 of the incoming array for the target value.
//It returns the index of the target value if is in the array
//If the target is not in the array, the function returns -1
{
   int mid, targetIndex=-1;
   int lowerBound=0;
   int upperBound=length-1;
   int found=false;  //the Java version would be boolean found=false;

   while (lowerBound<=upperBound && !found)
   {
      mid=(upperBound+lowerBound)/2;
      if (target<ar[mid])          //eliminate the upper half of this section
         upperBound=mid-1;
      else if (target>ar[mid])     //eliminate the lower half of this section
         lowerBound=mid+1;
      else                         //(target==ar[mid]) means it is found
      {
         targetIndex=mid;
         found=true;
      }
   }//end while
   return targetIndex;
}
```

```
//portion of C++ calling program:
   const int MAX=10;
   int ar[MAX], target, i, item;
   //get values
   cout<<'Enter '<<MAX<<' values separated by a space'<<endl;
   for (i=0; i<MAX; i++) cin>>ar[i];
   //get the target
   cout<<'What number should you search for in the array?';
   cin>>target;
   //perform the search
   item=BinarySearch(ar, MAX, target);
   if (item >=0) cout<<'The item is in location '<<item<<endl;
   else cout<<'The item is not in the array'<<endl;
```

Fig. 6-22 Binary search in C/C++.

```
while (lowerBound <= upperBound && !found)
```

Trace through the code:

ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]
2	4	6	7	9	10

target = 4	upper	lower	mid = (upper+lower)/2
set values	5	0	2
examine ar[2] ---- 4 < 6			
eliminate upper half	1	0	0
examine ar[0] ---- 4 > 2			
eliminate lower half	1	1	1
examine ar[1] ---- 4==4		FOUND	

target = 8	upper	lower	mid = (upper+lower)/2
set values	5	0	2
examine ar[2] ---- 8 > 6			
eliminate lower half	5	3	4
examine ar[4] ---- 8 < 9			
eliminate upper half	3	3	3
examine ar[1] ---- 8 > 7	lower == upper so NOT FOUND		

Fig. 6-23 Racing the binary search algorithm.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 6.4

6.4 TÌM KIẾM

Một trong những mục tiêu chính trong lập trình đó là tiết kiệm thời gian và công sức thông qua việc sử dụng lại mã. Một vài hàm tổng quát chẳng hạn như hàm tìm kiếm và phân loại các mảng có thể được viết theo một cách thức sao cho chúng hữu dụng trong nhiều ứng dụng khác nhau. Một số các thuật toán khác nhau để phân loại và tìm kiếm các mảng cũng được xác định và cũng có thể được thực thi trong bất kỳ ngôn ngữ nào. Hai thuật toán tìm kiếm và ba thuật toán phân loại sẽ được minh họa ở đây.

6.4.1 Tìm kiếm tự

Thuật toán đơn giản nhất để tìm kiếm một mảng đó là tìm kiếm theo trình tự. Trong phương pháp tìm kiếm theo trình tự một mảng phải được xem xét, một lần một phần tử cho đến khi giá trị đích được tìm thấy hoặc cho đến khi kết thúc mảng mà không tìm được giá trị đích. Phương pháp trình tự này thường là phương pháp chọn lựa tốt nhất khi rơi vào một trong hai trường hợp sau đây;

- *Kích thước mảng nhỏ hơn 100 phần tử.*
- *Các phần tử trong mảng không theo thứ tự đặc biệt.*

VÍ DỤ 6.25 *Hãy viết một hàm tìm kiếm tuầntự trong Visual Basic vốn nhận làm các tham số của mảng được tìm kiếm, chiều dài của nó và giá trị đích để tìm kiếm. Nó trả về số chỉ số ủa phần tử đang giữ giá trị đích hoặc trả về -1 nếu giá trị đích không nằm trong mảng. Hình 6.19 minh họa việc tgực thi hàm này.*

```
'Sequential Search Function
 Private Function SequentialSearch(ar() As Integer, length As Integer, _
                                   target As Integer) As Integer
'This function searches 1 through length of the incoming array for the target value.
'It returns the index of the element holding that value.
'If the target is not in the array, the function returns -1
  Dim lcv As Integer, targetIndex As Integer
  Dim found As Boolean
  found=False                  'set initial flag to false
  targetIndex=-1               'default value in case target is not found
  For lcv=1 To length
    If ar(lcv)=target Then
      found=True               'set flag to true when found
      targetIndex=lcv
    End If
 Next lcv
 SequentialSearch=targetIndex  'send back the target Index or -1
End Function
```

Hình 6.19 *Tìm kiếm theo tuần tự trong Visual Basic*

```
'portion of calling program
  const MAX=10
  Dim lcv As Integer            'loop control variable
  Dim response As Integer, target As Integer
  Dim scores(1 To MAX) As Integer
'read in scores
For lcv=1 To MAX
  scores(lcv)=InputBox("Enter the number")
Next lcv
'get the target value
target=InputBox("What number will you search for?")
response=SequentialSearch(scores, MAX, target)
If (response>=0) Then
   Print target; "is in the array at location"; response
Else
   Print target; "is not in the array"
End If
```

***Hình 6.19** (tiếp theo)*

Có nhiều lý do để thuật toán này không hữu dụng. Trước tiên nếu mảng này có hơn 100 phần tử, thì quy trình tìm kiếm phải mất rất nhiều thời gian. Thứ hai vòng lặp trong hàm tìm kiếm thông qua toàn bộ mảng, thậm chí nếu hạng mục được tìm thấy nằm ở index đầu tiên. Thứ ba, nếu đích nằm trong mảng nhiều lần thì hàm này chỉ trả về index cuối cùng của nó.


```
'alternate form of loop for the sequential search function
Private Function SequentialSearch(ar() As Integer, length As Integer, _
        ByVal target As Integer) As Integer
'This function searches 1 through length of the incoming array for the target value.
'It returns the index of the element holding that value.
'If the target is not in the array, the function returns -1
  Dim lcv As Integer, targetIndex As Integer
  Dim found As Boolean
  found=False                   'set initial flag to false
  targetIndex=-1
  lcv=1
  Do While (lcv<=length And (Not found))   'use do while to stop when item is found
    If ar(lcv)=target Then
      found=True
      targetIndex=lcv
    End If
    lcv=lcv+1
  Loop
  SequentialSearch=targetIndex 'send back the target index or -1
End Function
```

***Hình 6.20** Phương pháp tìm kiếm tuần tự hiệu quả hơn*

***VÍ DỤ 6.26** Một sự điều chỉnh nhỏ về vòng lặp trong hàm, được minh họa ở hình 6.20 bảo đảm nó ngưng ngay khi giá trị đích được tìm thấy. Thay vì vòng lặp For... Next sẽ thực thi cho đến khi đạt đến chiều dài, thì một vòng lặp Do While được dùng nó ngưng ngay khi hạng mục được tìm thấy. Trong ví dụ này nếu đích nằm trong mảng*

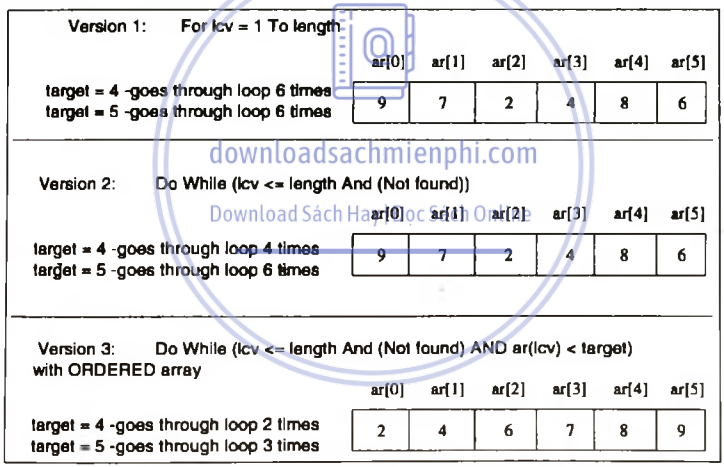
nhiều lần thì hàm số này chỉ trả về chỉ mục FIRST. Một hàm tìm kiếm khác sẽ trả về số các lần mà hạng mục có được trong vòng lặp được như minh họa trong bài tập có lời giải 6.10 ở cuối chương này.

VÍ DỤ 6.27 *Có một phương pháp chỉnh sửa được thực hiện trong việc tìm kiếm tuần tự ví dụ trước đây để gia tăng hiệu quả nếu đích không nằm trong mảng.*

Nếu mảng chuyển tới theo thứ tự tăng dần thì khi một phần tử được tìm thấy lớn hơn đích, thì mảng có thể ngưng. Điều này yêu cầu rằng chỉ có một dòng bị thay đổi, câu lệnh Do While như sau:

```
Do While (lcv<=length And (Not found) AND ar(lcv) <target)
```

Hình 6.21 minh họa số các lần mà vòng lặp sẽ thực thi ứng với mỗi thuật toán tìm kiếm tuần tự dùng cho mục tiêu trong mảng và một mục tiêu không nằm trong mảng.



Hình 6.21 *So sánh các điều kiện thử nghiệm tìm kiếm tuần tự.*

6.4.2 Tìm kiếm nhị phân

Phần phiên bản chỉnh sửa sau cùng rõ ràng là một cải tiến đáng kể nhất trong quy trình tìm kiếm tuần tự ùng cho một mảng có thứ tự. Tuy nhiên thậm thuật toán đó cũng phải mất một khoảng thời gian dài để tìm kiếm một mảng ớn chẳng hạn như một mảng có 10 ngàn phần tử. Một giải pháp tốt hơn để tìm một giá trị đích trong một mảng và sắp xếp thứ tự mà không có sự trùng lặp đó là sử dụng thuật toán tìm kiếm nhị phân.

Trong thuật toán tìm kiếm nhị phân cho nên một nửa toàn bộ đầu tiên của danh sách có thể bị bỏ qua.

- *Nếu phần tử đó là đích đang được tìm kiếm thì chỉ số ở giữa được cho ra.*
- *Nếu phần tử giữa nhỏ hơn phần tử đích thì theo thứ tự toàn bộ nửa đanh sách đầu có thể bị bỏ qua.*
- *Nếu phần tử giữa lớn hơn phần tử đích, thì toàn bộ nửa sau của danh sách bị bỏ qua*

Các phần của mảng được cắt một nửa thành công, loại bỏ số các phần tử lớn đến khi đích được tìm thấy hoặc cho đến khi chỉ có một phần tử còn trong mục được xem xét.

Tìm kiếm nhị phân thường là phần chọn lựa tốt nhất nếu kích cỡ của mảng lớn. Các phần tử trong mảng thì được xếp theo thứ tự, vì vậy cần phải có khoảng thời gian cần thiết để trước tiên phân loại mảng theo thứ tự. Phép phân loại sẽ được xem xét trong phần dưới đây.

VÍ DỤ 6.28 *Hãy viết mã cần thiết để thực thi hàm tìm kiếm nhị phân trong C/C++. Hàm này nhận các tham số giống hệt như tìm kiếm tuần tự: mảng, chiều dài mảng với giá trị đích.*

Hình 6-22 minh họa hàm này. Phiên bản Java của hàm này cũng giống như vậy ngoại trừ có một dường được đánh dấu.

```
//C/C++ version of Binary Search (with Java change)
int BinarySearch(int ar[], int length, int target)
//This function searches 0 through length-1 of the incoming array for the target value.
//It returns the index of the target value if is in the array
//If the target is not in the array, the function returns -1
{
    int mid, targetIndex=-1;
    int lowerBound=0;
    int upperBound=length-1;
    int found=false;  //the Java version would be boolean found=false;

    while (lowerBound<=upperBound && !found)
    {
        mid=(upperBound+lowerBound)/2;
        if (target<ar[mid])          //eliminate the upper half of this section
            upperBound=mid-1;
        else if (target>ar[mid])     //eliminate the lower half of this section
            lowerBound=mid+1;
        else                         //(target==ar[mid]) means it is found
        {
```

Hình 6.22 *Tìm kiếm nhị phân trong C/C++*

```
            targetIndex=mid;
            found=true;
        }
    }//end while
    return targetIndex;
}
//portion of C++ calling program:
    const int MAX=10;
    int ar[MAX], target, i, item;
    //get values
    cout<<'Enter '<<MAX<<' values separated by a space'<<endl;
    for (i=0; i<MAX; i++) cin>>ar[i];
    //get the target
    cout<<'What number should you search for in the array?';
    cin>>target;
    //perform the search
    item=BinarySearch(ar, MAX, target);
    if (item >=0) cout<<'The item is in location '<<item<<endl;
    else cout<<'The item is not in the array'<<endl;
```

Hình 6.22 *(tiếp theo)*

Hình 6-23 trình bày được thực thi mã cho hai giá trị đích khác nhau. Nếu mảng này nhỏ thì thời gian tiết kiệm không nhiều lắm. Tuy nhiên, nếu mảng này có 1000 hạng mục thì vấn đề xem xét đầu tiên đó là loại bỏ một nửa hạng mục. Điều này tiết kiệm được một lượng thời gian đáng kể. Phương pháp tìm kiếm nhị phân là một phương pháp tìm kiếm rất nhanh, nhưng mảng phải được phân loại thứ tự. Nếu mảng không theo thứ tự, thì nó sẽ được phân loại trước khi áp dụng phương pháp tìm kiếm nhị phân. Còn nếu không thể được phân loại thì quy trình tìm kiếm tuần tự chính là chọn lựa duy nhất.

while (lowerBound <= upperBound && !found)

Trace through the code:

ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]
2	4	6	7	9	10

target = 4	upper	lower	mid = (upper+lower)/2
set values	5	0	2
examine ar[2] ---- 4 < 6			
eliminate upper half	1	0	0
examine ar[0] ---- 4 > 2			
eliminate lower half	1	1	1
examine ar[1] --- 4==4		FOUND	

Hình 6.23 *Vẽ sơ đồ thuật toán tìm kiếm nhị phân*

target = 8	upper	lower	mid = (upper+lower)/2
set values	5	0	2
examine ar[2] ---- 8 > 6			
eliminate lower half	5	3	4
examine ar[4] ---- 8 < 9			
eliminate upper half	3	3	3
examine ar[1] ---- 8 > 7	lower == upper so NOT FOUND		

Hình 6.23 (tiếp theo)

CHỦ ĐIỂM 6.5

SORTING
Sắp xếp thứ tự

Both the binary search and the most efficient of the sequential search algorithms require that the array be sorted in ascending order. Many applications also depend upon keeping various lists in numeric or alphabetical order. Many algorithms for sorting arrays have been designed, some of which are very complex. Three sorting algorithms will be explained in this section: bubble sort, selection sort, and insertion sort. These are often not the most efficient sorting algorithms, but are usually the easiest to understand and code.

6.5.1 Selection Sort - *Sắp xếp theo phương pháp chọn trực tiếp*

The concept of the selection sort algorithm is fairly simple. Successive passes are made through the array to find the largest element, and then put it in its proper place. The algorithm is:

(1) Pass 1: Examine the entire array and find the largest element.
(2) Place it in its proper spot at the end of the array.
(3) Pass 2: Examine the entire array except for the last element (already placed) and find the next largest element.
(4) Place the next largest in the spot second from the end.
(5) Continue the passes until there is only one element left. The last pass is not necessary because the last item left must be the smallest and already at the beginning of the array.

In each pass through the array all the elements are examined from the beginning until reaching the point where an element has already been placed in its proper spot.

EXAMPLE 6.29 Trace the algorithm and show how it would sort an array of characters (S, A, N, Z, B) and then write a Visual Basic subprogram that performs the selection sort on an array of strings.

Figure 6-24 illustrates the tracing of the selection sort algorithm. Figure 6-25 shows the Visual Basic code.

In the Visual Basic code, each pass through the array is performed by the outer loop which executes length-1 times. The inner loop examines each element in the array from the beginning to the limit of this particular pass to find the largest. After the inner loop stops, the largest element (maxIndex) is swapped with the item in the last spot of this pass (limit). In this particular algorithm, if maxIndex and limit are the same spot, it swaps

them anyhow. This could be avoided by using an If statement to test whether the largest element is already in its proper spot:

```
If (maxIndex<>limit) Then
   temp=ar(limit)
   ar(limit)=ar(maxIndex)
   ar(maxIndex)=temp
End If
```

6.5.2 Bubble Sort - *Sắp xếp theo kiểu nổi bọt*

The algorithm for the bubble sort also makes use of successive passes through the array. However, instead of searching for one largest element, the bubble sort compares each successive pair of elements and places the larger one below the other. The largest item in the array will "bubble" down to the end on the first pass. The algorithm is:

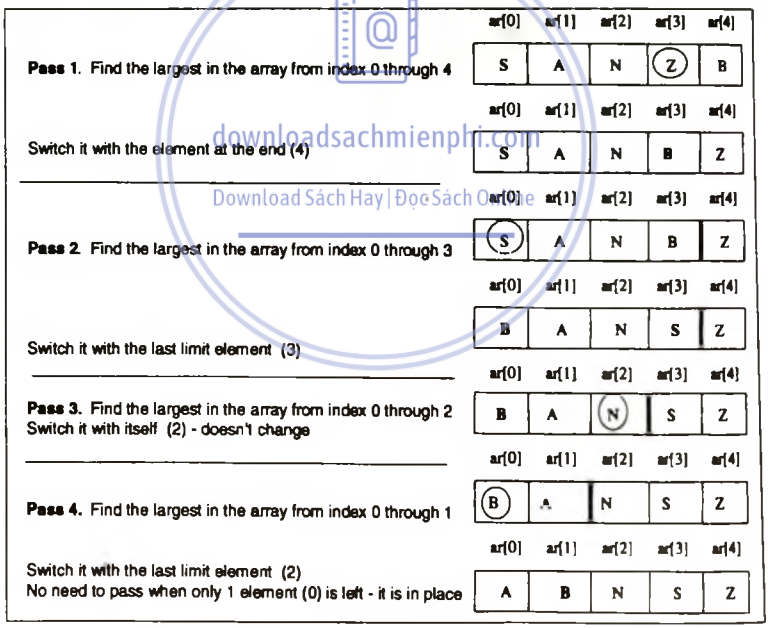


Fig. 6-24 Racing the selection sort algorithm.

```
Sub SelectionSort(ByRef ar() As String, ByVal length As Integer)
'.....................................................................
'Takes in an array of integers and the length of the list
'Returns list sorted into ascending order
'.....................................................................
 Dim maxIndex As Integer          'Index of largest num in each pass
 Dim limit As Integer             ''limit" for each pass - last item not placed
 Dim i As Integer
 Dim temp As String
 For limit=length-1 To 1 Step-1 'begin each pass
   maxIndex=0
   For i=1 To limit               'in this pass, examine each element
     If ar(i)>ar(maxIndex) Then
       maxIndex=i                 'find the maximum
     End If
   Next i
   temp=ar(limit)                 'swap the largest with the current limit
   ar(limit)=ar(maxIndex)
   ar(maxIndex)=temp
 Next limit
End Sub
```

Fig. 6-25 Visual Basic selection sort.

(1) Pass 1: Compare each pair of items first through last and place the larger element second.

(2) Pass 2: Compare each pair of items first through second from last and put the larger item second.

(3) Continue the passes until there is only one element left. The last pass is not necessary because the last item left must be the smallest and already at the beginning of the array.

In each pass through the array all the elements are examined from the beginning until reaching the point where an element has already been placed in its proper spot.

EXAMPLE 6.30 Trace through the bubble sort algorithm and show how it sorts an array of integers (4, 8, 2, 0, 6). Then write a C++ function to perform this sort.

Figure 6-26 traces through the algorithm and illustrates how it sorts an array of integers. Figure 6-27 shows a C++ function that performs the bubble sort on an array of integers.

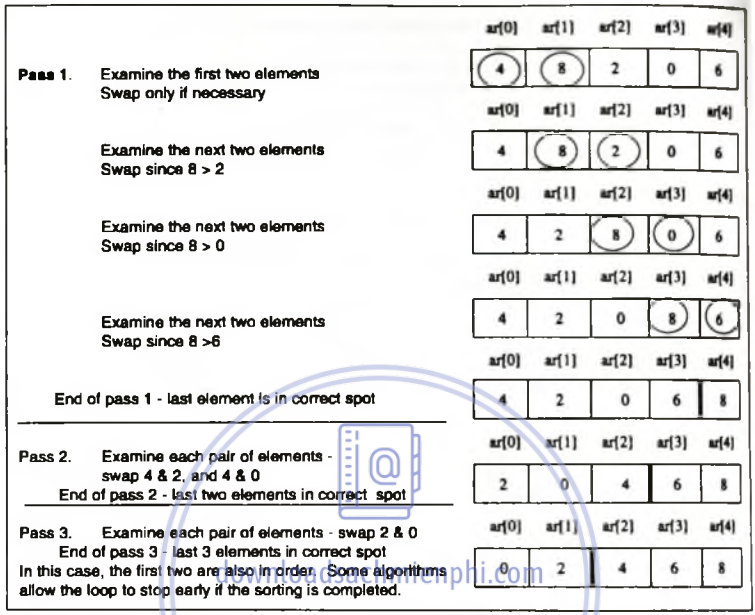


Fig. 6-26 Racing the bubble sort algorithm.

Notice that the code above uses a more modular approach to sorting by using a separate function to perform the swap. This function can be called from any sort function. Since the incoming parameters are being switched, it is important that they come into the function as call by reference parameters.

In the C++ code, each pass examines all the pairs of elements up to the limit where elements are already placed in their correct spot, even if the items are already in order. For arrays that might come into the function already sorted, this would take an unnecessary amount of time.

EXAMPLE 6.31 It is possible to speed up the sorting, especially for lists that are already mostly sorted, by reducing the number of passes. If the array is sorted, then no swaps are necessary. The Java function in Fig. 6-28 demonstrates the more efficient version of the bubble sort.

In the Java code of Fig. 6-28, if the array is already in order, only one pass is made with no swaps and the loop ends. There is no way to make the selection sort this efficient, so if the array comes in mostly sorted, the bubble sort is more efficient.

```
// swap function can be used by several sorts
void Swap(int& num1,int& num2 )
  //.............................................
  //Takes in 2 numbers and swaps their contents
  //.............................................
{
  int temp=num1;
  num1=num2;
  num2=temp;
}
void BubbleSort( int ar[], int length )
  //.....................................................
  //Takes in an array of integers and the length of the list
  //Returns list sorted into ascending order
  //.....................................................
{
  int limit;                    //'limit' for each pass
  int i;
  limit=length-1;
  while (limit>0) {             //each pass through the array
    for (i=0; i<limit; i++)     // successive comparisons in this pass
      if (ar[i]>ar[i+1])
          Swap(ar[i], ar[i+1]); // calls swap function above
      limit--;
  }
}
```


Fig. 6-27 C/C++ bubble sort.

6.5.3 Insertion Sort - Sắp xếp theo phương pháp chèn trực tiếp

The bubble sort can be coded to recognize an already sorted list and stop before executing all the length-1 passes. However, if even only a few elements are out of order, it could take several passes and a large number of comparisons to perform the sort. The insertion sort algorithm takes greater advantage of the partial ordering of an array. The algorithm for the insertion sort is:

(1) Pass 1: Examine the first two elements and place them in the proper order with respect to each other.

(2) Pass 2: Starting at the third element, find its correct spot relative to the first two. If necessary, move the elements above down accordingly.

(3) Continue the passes starting with the next element, the limit element, until the last element is placed in its proper spot.

```
void BubbleSort( int list[], int length )
  //.....................................................
  // Takes in an array of integers and the length of the list
  // Returns list sorted into ascending order
  //.....................................................
{
  int limit;            //"limit" for each pass
  int i;
  boolean swaps=true;  //boolean variable to see if there were swaps
  limit=length-1;
  while (limit>0 && swaps)
  {
    swaps=false;        //set initially to false
    for (i=0; i<limit; i++)
      if (list[i]>list[i+1])
      {
          int temp=list[i];
          list[i]=list[i+1];
          list[i+1]=temp;
          swaps=true;   //a swap occurred in this pass
      }
    limit--;            //if no swaps occurred, swaps is still false to stop loop
  }
}
```


Fig. 6-28 Java version of more efficient bubble sort.

The bubble and selection sort place the largest items in their spots first. The insertion sort does not look for the largest elements. Instead, it moves the others down looking for the place to insert the next item until the correct spot is found. After EACH pass, all the elements that have been considered so far are in their proper order.

EXAMPLE 6.32 Trace through the insertion sort algorithm as it sorts an array of strings ("Tom", "Joe", "Sue", "Low", "Ann"). Then write a Java function to implement the insertion sort.

Figure 6-29 traces through the algorithm and illustrates how it sorts the array of strings. Figure 6-30 shows a Java function that performs the insertion sort on an array of strings.

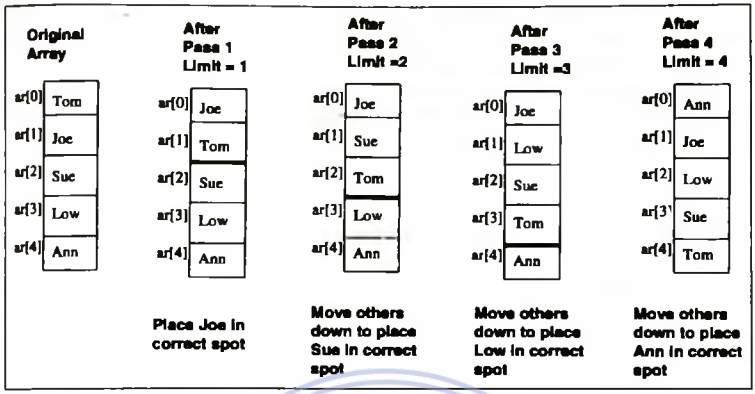


Fig. 6-29 Tracing the insertion sort algorithm.

```
void InsertionSort( String list[], int length )
  //.....................................................
  //Takes in an array of Strings and the length of the list
  //Returns list sorted into ascending order
  //.....................................................
{
  String itemToInsert;        //item to insert in each pass
  boolean stillLooking;       //Boolean variable to control loop
  int limit, j;
  for (limit=1; limit<length; limit++)
  {
       //walk backwards through the list looking for slot to insert list[i]
       itemToInsert=list[limit];
       j=limit-1;
       stillLooking=true;
       while (j>=0 && stillLooking)
       {
          if (itemToInsert.compareTo(list[j])<0) //<0 if 1st is greater than 2nd
          {
             list[j+1]=list[j];
             j--;
          }
          else stillLooking=false;
       }//end while
       list[j+1]=itemToInsert;
   }//end for
}
```

Fig. 6-30 Java insertion sort.

The insertion sort performs the least amount of comparisons if the array comes into the array already partially sorted. This function also demonstrates another comparison operator for Java Strings. The Stringl.compareTo(String2) returns a 0 if the strings are equal, a negative number if Stringl is less than String2 and a positive number if Stringl is greater than String2.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 6.2

6.5 SẮP XẾP

Cả hai thuật toán dùng cho tìm kiếm nhị phân và tìm kiếm tuần tự được yêu cầu rằng mảng phải được sắp xếp theo thứ tự tăng dần. Nhiều chương trình ứng dụng cũng phụ thuộc vào việc giữ các danh sách khác nhau theo thứ tự bản chữ cái hoặc thứ tự số. Nhiều thuật toán dùng để sắp xếp mảng phải được thiết kế. Một số thuật toán rất là phức tạp. Ba trong số các thuật toán sắp xếp phải được giải thích trong phần này, đó là sắp xếp theo bong bóng, sắp xếp theo nổi bọt, sắp xếp theo chọn lựa và sắp xếp theo chèn. Đây không phải là những thuật toán sắp xếp hiệu quả nhất nhưng lại là phương pháp dễ hiểu nhất và dễ tạo mã.

6.5.1 Sắp xếp theo phương pháp chọn trực tiếp

Khái niệm về thuật toán phân loại chọn hoàn toàn đơn giản. Quy trình truyền thành công được thực hiện thông qua mảng để tìm phần tử lớn nhất rồi đưa nó vào ở vị trí phù hợp. Thuật toán là:

(1) Bước 1: Sao chép toàn bộ mảng và tìm phần tử lớn nhất.

(2) Đặt nó vào một điểm phù hợp ở cuối mảng.

(3) Bước 2: xem xét toàn bộ mảng ngoại trừ phần tử sau cùng (tức là phần tử đã được đặt vào vị trí rồi) rồi tìm phần tử lớn nhất kế tiếp.

(4) Đặt phần tử lớn nhất kế tiếp vào vị trí thứ hai tính từ cuối.

(5) Tiếp tục các bước còn lại cho đến khi chỉ còn lại một phần tử. Bước cuối cùng không cần thiết bởi vì phần tử còn lại cuối cùng này là phần tử nhỏ nhất và đã nằm ở đầu mảng rồi.

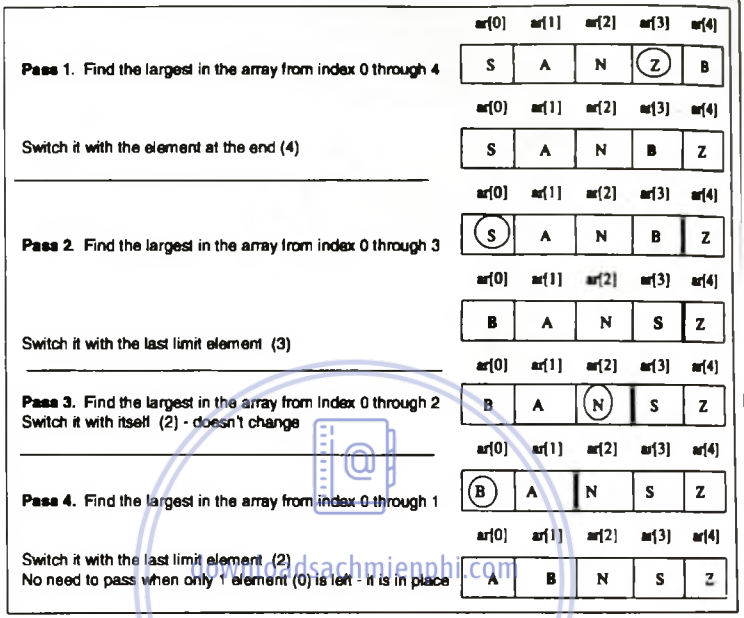
Trong mỗi bước thông qua mảng, tất cả các phần tử phải được xem xét từ ban đầu cho đến khi đạt đến điểm nơi mà phần tử đã được đặt ở vị trí hoàn hảo của nó.

```
//C/C++ version of Binary Search (with Java change)
int BinarySearch(int ar[], int length, int target)
//This function searches 0 through length-1 of the incoming array for the target value
//It returns the index of the target value if is in the array
//If the target is not in the array, the function returns -1
{
   int mid, targetIndex=-1;
   int lowerBound=0;
   int upperBound=length-1;
   int found=false;  //the Java version would be boolean found=false;

   while (lowerBound<=upperBound && !found)
   {
      mid=(upperBound+lowerBound)/2;
      if (target<ar[mid])          //eliminate the upper half of this section
         upperBound=mid-1;
      else if (target>ar[mid])     //eliminate the lower half of this section
         lowerBound=mid+1;
      else                         //(target==ar[mid]) means it is found
      {
         targetIndex=mid;
         found=true;
      }
   }//end while
   return targetIndex;
}
//portion of C++ calling program:
   const int MAX=10;
   int ar[MAX], target, i, item;
   //get values
   cout<<"Enter "<<MAX<<" values separated by a space"<<endl;
   for (i=0; i<MAX; i++) cin>>ar[i];
   //get the target
   cout<<"What number should you search for in the array?";
   cin>>target;
   //perform the search
   item=BinarySearch(ar, MAX, target);
   if (item >=0) cout<<"The item is in location "<<item<<endl;
   else cout<<"The item is not in the array"<<endl;
```


***Hình 6-22**. Tìm kiếm nhị phân trong C/C++*

***VÍ DỤ 6-22** Vẽ thuật toán và chứng minh cách mà nó săp xếp một mảng các ký tự {S, A, N, Z. B} và viết một chương trình con Visual Basic thực thi phép sắp xếp chọn trên một mảng các chuỗi.*

Hình 6-24 minh họa sơ đồ thuật toán sắp xếp theo phương pháp chọn trực tiếp. Hình 6-25 minh họa mảng Visual Basic



Hình 6-24. *Sơ đồ thuật toán phân loại chọn*

```
Sub SelectionSort(ByRef ar() As String, ByVal length As Integer)
'.....................................................................
'Takes in an array of integers and the length of the list
'Returns list sorted into ascending order
'.....................................................................
  Dim maxIndex As Integer          'Index of largest num in each pass
  Dim limit As Integer             'limit' for each pass - last item not placed
  Dim i As Integer
  Dim temp As String
  For limit=length-1 To 1 Step-1 'begin each pass
    maxIndex=0
    For i=1 To limit               'in this pass, examine each element
      If ar(i)>ar(maxIndex) Then
        maxIndex=i                 'find the maximum
      End If
    Next i
    temp=ar(limit)                 'swap the largest with the current limit
    ar(limit)=ar(maxIndex)
    ar(maxIndex)=temp
  Next limit
End Sub
```

Hình 6-25. *Phép phân loại Visual Basic*

Trong mã Visual Basic mỗi bước thông qua quy trình tạo mảng được thực thi bởi phép vòng lập ngoài, nó thực thi toàn bộ chiều dài -1 lần. Vòng lặp bên trong xem xét một phần tử trong mảng từ đầu cho đến giới hạn của lần tìm đặc biệt này để tìm giá trị lớn nhất. Sau khi vòng lập trong ngưng, thì phần tử lớn nhất (maxIndex) được hoán chuyển với hạng mục nằm ở điểm cuối cùng của lần này (giới hạn) trong thuật toán đặc biệt này nếu maxIndex giới hạn nằm cùng một điểm giống nhau, thì nó cũng tự hoán chuyển cho nhau cho bất cứ hình thức nào. Chúng ta có thể tránh được điều này bằng cách sử dụng một câu lệnh If để thử nghiệm phần tử lớn nhất xem nó có nằm đúng vị trí riêng biệt hay không.

```
If (maxIndex<>limit) Then
   temp=ar(limit)
   ar(limit)=ar(maxIndex)
   ar(maxIndex)=temp
End If
```

6.5.2 Sắp xếp theo phương pháp nổi bọt

Thuật toán để sắp xếp nổi bọt cũng sử dụng các bước tuần tự thông qua mạng. Tuy nhiên, thay vì tìm kiếm một phần tử lớn nhất, phép sắp xếp nổi bọt so sánh mỗi một cặp phần tử tuần tự và đặt phần tử lớn hơn nằm bên dưới phần tử khác. Hạng mục lớn nhất trong mảng sẽ là "nổi bọt" bên dưới đến kiểu cuối của bước đầu tiên. Thuật toán là:

(1) Bước 1: So sánh các cặp phần tử đi từ đầu cho đến cuối rồi đặt phần tử lớn hơn ra sau.

(2) Bước 2: So sánh mỗi cặp phần tử từ đầu cho đến từ ký tự hai tính từ cuối và đặt phần tử lớn hơn vào vị trí hai.

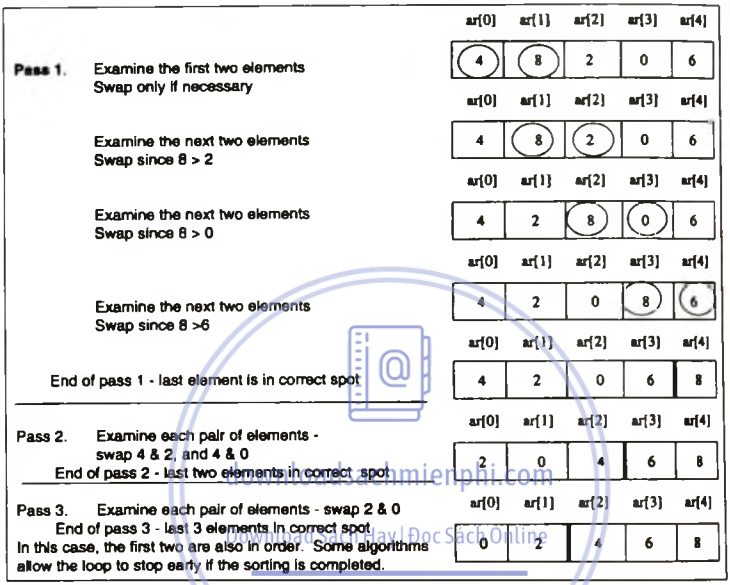
(3) Tiếp tục các bước cho đến khi chỉ còn một phần tử duy nhất. Bước cuối cùng không cần thiết bởi vì hạng mục sau còn lại phải là hạng mục nhỏ nhất và đã nằm ở đầu mảng rồi.

Trong mỗi bước thông qua quy trình tạo mảng tất cả các phần tử đều được xem xét từ ban đầu cho đến khi đạt được điểm nơi mà một phần tử đã được đặt đúng điểm hoàn chỉnh của nó.

***VÍ DỤ 6-30** Vẽ sơ đồ thuật toán sắp xếp theo bọt và chỉ cách mà nó sắp xếp một mảng các số nguyên (4, 8,2, 0, 6) sau đó viết một hàm số C++ để thực hiện bước phân loại này.*

Hình 6-26 trình bày thuật toán và minh họa mà nó sắp xếp một mảng các số nguyên.

Hình 6-27 trình bày một hàm C++ thực thi phép phân loại bọt bóng trên một mảng các số nguyên.



Hình 6-26. *Sa đồ thuật toán phân loại bọt*

Lưu ý rằng mã trên đây đã sử dụng một phương pháp tạo module để phân loại bằng cách sử dụng một hàm riêng biệt để thực hiện phép hoán chuyển. Hàm này có thể được gọi từ bất cứ hàm phân loại nào. Bởi vì các tham số đưa vào đang được hoán chuyển, nên điều quan trọng là chúng phải đưa vào hàm dưới dạng cuộc gọi bởi các tham số chiếu.

Trong mã C++, mỗi bước xem xét tất cả các cặp phần tử phải tiến lên đến giới hạn nơi mà các phần tử đã được đặt ở vị trí hoàn chỉnh của nó, thậm chí nếu các phần tử đã có sẵn thứ tự rồi. Đối với các mảng được đưa vào trong hàm phân loại sẵn thì điều này mất một lượng thời gian không cần thiết.

VÍ DỤ 6-31 *Ta có thể tăng tốc độ phân loại đặc biệt đối với các danh sách đã được phân loại sẵn bằng cách giảm số các bước nếu mảng đã được phân loại thì không cần thiết phải hoán chuyển nữa. Hàm Java trong hình 6-28 minh hoạ phiên bản hiệu quả của phép phân loại bọt.*

```
// swap function can be used by several sorts
void Swap(int& num1,int& num2 )
  //.............................................
  //Takes in 2 numbers and swaps their contents
  //.............................................
{
  int temp=num1;
  num1=num2;
  num2=temp;
}
void BubbleSort( int ar[], int length )
  //.....................................................
  //Takes in an array of integers and the length of the list
  //Returns list sorted into ascending order
  //.....................................................
{
  int limit;                    //"limit" for each pass
  int i;
  limit=length-1;
  while (limit>0) {             //each pass through the array
    for (i=0; i<limit; i++)     // successive comparisons in this pass
      if (ar[i]>ar[i+1])
         Swap(ar[i], ar[i+1]);  // calls swap function above
      limit--;
  }
}
```


Hình 6-27. *Phân loại bọt C/C++*

Trong mã Java của hình 6- 28 nếu mảng này đã được sắp xếp thứ tự rồi thì chỉ cần một bước thực hiện mà không có sự hoán chuyển nào mà vòng lặp kết thúc. Không có cách nào để làm cho phép sắp xếp theo phương pháp chọn trực tiếp. Vì thế nếu mảng đã được sắp xếp xong thì phép phân loại bọt đã được hiệu quả hơn.

6.5.3 Sắp xếp thao phương pháp chèn trực tiếp

Phép sắp xếp theo phương pháp nổi bọt có thể được tạo mã để nhận biết một danh sách đã được sắp xếp thứ tự rồi và ngưng trước khi thực thi tất cả chiều dài từ một bước. Tuy nhiên, thậm chí chỉ khi có một ít phần tử không nằm theo thứ tự thì phải mất nhiều bước và phải thực hiện một số lớn phép so sánh để thực hiện phép phân loại. Thuật toán này có nhiều ưu điểm hơn đối với thứ tự từng phần của một mảng. Thuật toán dùng cho phép phân loại chèn là:

(1) Bước 1: hai phần tử đầu tiên họ đặt chúng theo thứ tự hoàn chỉnh tương ứng với nhau.

```
void BubbleSort( int list[], int length )
  //.....................................................
  // Takes in an array of integers and the length of the list
  // Returns list sorted into ascending order
  //.....................................................
{
  int limit;              //"limit" for each pass
  int i;
  boolean swaps=true;  //boolean variable to see if there were swaps
  limit=length-1;
  while (limit>0 && swaps)
  {
    swaps=false;          //set initially to false
    for (i=0; i<limit; i++)
      if (list[i]>list[i+1])
      {
          int temp=list[i];
          list[i]=list[i+1];
          list[i+1]=temp;
          swaps=true;     //a swap occurred in this pass
      }
    limit--;              //if no swaps occurred, swaps is still false to stop loop
  }
}
```

***Hình 6-28**. Minh hoạ phiên bản hiệu quả của phép sắp xếp theo phương pháp nổi bọt mã Java.*

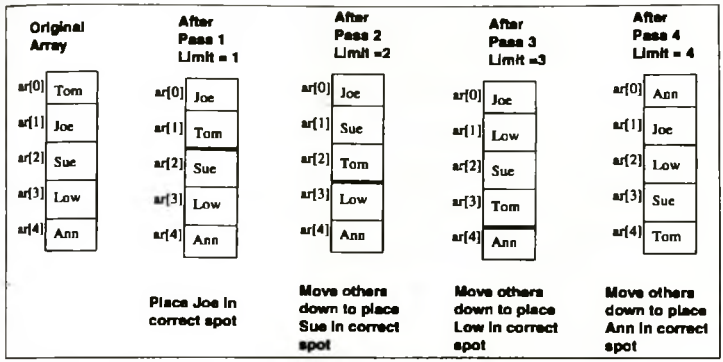
(2) Bước 2: bắt đầu tại phần tử thứ ba, tìm vị trí đúng của nó tương ứng với hai phần tử đầu tiên nếu cần hãy di chuyển các phần tử lên trên hoặc xuống dưới cho phù hợp.

(3) Tiếp tục các bước bắt đầu với phần tử kế tiếp, phần tử giới hạn, cho đến khi phần tử sau cùng được đặt ở điểm hoàn chỉnh của nó.

Phép phân loại bọt và phân loại chọn đặt các hạng mục lớn nhất ở các vị trí hoàn chỉnh của nó trước tiên. Còn phép phân loại chèn thì không xem xét các phần tử lớn nhất, thay vào đó nó di chuyển các phần tử xuống để tìm kiếm chỗ chèn hạng mục kế tiếp cho đến khi vị trí được tìm thấy, sau mỗi bước, tất cả các phần tử đều được xem ở thứ tự hoàn chỉnh của chúng.

***VÍ DỤ 6-32** Hãy vẽ sơ đồ thông qua thuật toán phân loại chèn khi nó phân loại một mảng các chuỗi ["Tom", "Joe", "Sue", "LOw", "Ann"]. Sau đó viết một hàm Java để thực thi phép phân loại chèn này.*

Hình 6-29 trình bày sơ đồ về thuật toán và minh họa cách mà nó phân loại mảng các chuỗi. Hình 6-30 minh họa một hàm Java thực hiện phép phân loại chèn trên một mảng các chuỗi.



Hình 6-29. *Vẽ sơ đồ thuật toán phép phân loại chèn*

```
void InsertionSort( String list[], int length )
  //.....................................................
  //Takes in an array of Strings and the length of the list
  //Returns list sorted into ascending order
  //.....................................................
{
  String itemToInsert;        //item to insert in each pass
  boolean stillLooking;       //Boolean variable to control loop
  int limit, j;
  for (limit=1; limit<length; limit++)
  {
      //walk backwards through the list looking for slot to insert list[i]
      itemToInsert=list[limit];
      j=limit-1;
      stillLooking=true;
      while (j>=0 && stillLooking)
      {
          if (itemToInsert.compareTo(list[j])<0) //<0 if 1st is greater than 2nd
          {
              list[j+1]=list[j];
              j--;
          }
          else stillLooking=false;
      }//end while
      list[j+1]=itemToInsert;
   }//end for
}
```

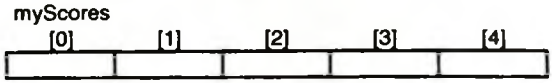
Hình 6-30. *Phép phân loại chèn Java*

Phép phân loại chèn thực hiện lượng so sánh tối thiểu nhất nếu mảng này đã trở thành mảng đã phân loại từng phần rồi, hàm này cũng minh họa một toán tử so sánh khác dùng cho các chuỗi Java. String1.compareTo(string2) trả về một 0 nếu chuỗi bằng nhau, trả về một số âm nếu String1 nhỏ hơn string2 và trả về một số dương nếu String1 lớn hơn String2.

SOLVED PROBLEMS
Bài tập có lời giải

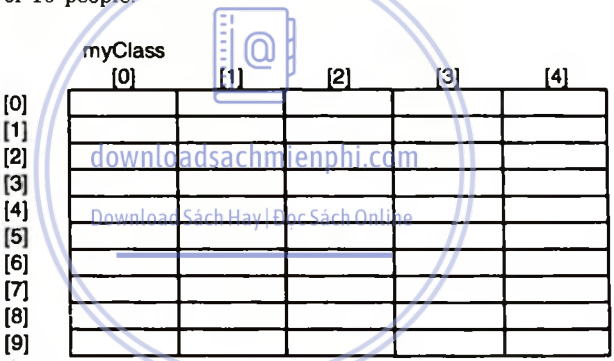
6.1 Draw a picture of the following arrays:

(a) An array called myScores to keep track of 5 tests scores.



Arrays can be pictured either vertically or horizontally. In the computer the elements are consecutive memory locations.

(b) An array called myClass to keep track of 5 test scores for a class of 10 people.



6.2 Given an array called myArray declared with 10 items, or boxes from [0] to [9], which of the following designations would be legal calls? Assume X = 5.

(a) `myArray [X]`

(b) `myArray [2*X]`

(c) `myArray [ 0 ]`

(d) `myArray [X-6]`

(e) `myArray [4*X]`

(a) Legal because there is a box 5.

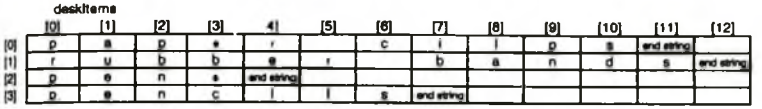
(b) Illegal because there is no box 10.

(c) Legal because there is a box 0.

(d) Illegal because there is no box -1.

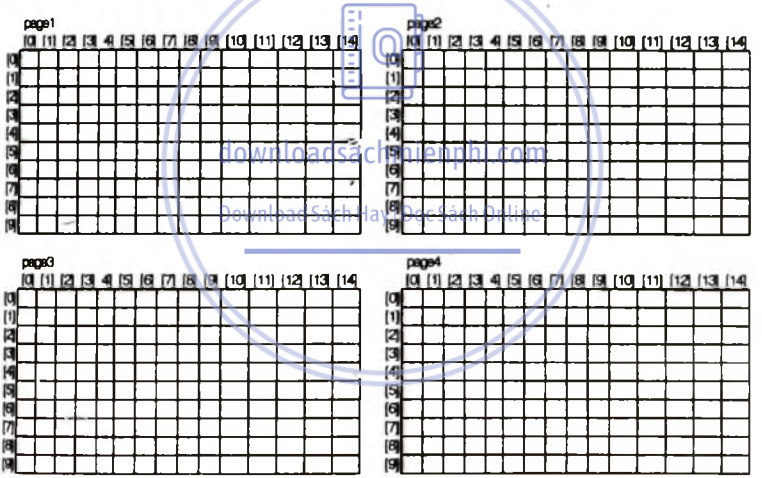
(e) Illegal because there is no box 20.

6.3 Draw a picture of an array called *deskItems* that contains a list of items usually found on a desk. The items are: paper clips, rubber bands, pens, and pencils.



Notice in this example that a space, as in row 0 column 5, is a character. Also, if some strings are short like "pens," and some are long like "rubber bands," the array must be large enough for the largest string even if memory locations are wasted following the shortest element.

3.4 Draw a picture of a three-dimensional array called *book* that contains 4 pages with 10 lines of 15 integers on each page.



6.5 Write the Visual Basic, C/C++, and Java code to declare the three-dimensional array in the previous problem.

(a) VB:
```
Dim book (4, 10, 15) As Integer
```

(b) C/C++:
```
int book [ 4 ] [ 10 ] [ 15 ] ;
```

(c) Java:
```
int book [ ] [ ] [ ] ;
book=new int[4][10][15];
```

6.6 Write the Visual Basic declaration statements for both arrays in Solved Problem 6.1 above.

(a) `Dim myScores (5) As Integer` or `Dim myScores (0 to 4) As Integer`

(b) `Dim myClass (10 , 5) As Integer` or `Dim myClas's (0 to 9 , 0 to 4) As Integer`

In Visual Basic, if the lower subscript is not given, it is assumed to be 0. The two Dim statements in each part produce identical results.

6.7 Write the C/C++ declaration statements for both arrays in Solved Problem 6.1 above.

(a)
```
int myScores [5];      //C/C++ uses the brackets instead
                          of parentheses
```

(b)
```
int myClass [10][5]; //C/C++ defines the rows and col
                        umns in separate brackets
```

6.8 Write the Java declaration statements for both arrays in Solved Problem 6.1 above.

(a)
```
int myScores [ ];      //declare the array
myScores=new int       [5];  //allocate the memory
                             space
```

(b)
```
int myClass [ ][       ];    //declare the array
myClass=new int [10][5];     //allocate the memory
                             space
```


6.9 Write the Visual Basic, C/C++, and Java declaration statements for the string array in Solved Problem 6.3. For C++ and Java, assign the values when memory is allocated.

(a) VB: `Dim deskItems (1 To 4) As String`

(b) *C/C++:*
```
char *deskItems [4 ] ={"paper clips", "rubber
bands", "pens", "pencils"};
```

(c) Java:
```
String deskItems[]=new String[4];
deskItems[0]="paper clips";
deskItems[1]="rubber bands";
deskItems[2]="pens"; deskItems[3]="pencils";
```

Remember, in C and C+ + there is no built-in String type in most compilers. The structure must be an array of pointers to strings.

6.10 Write a sequential search function that returns the number of times the item is present in the array, not the index of the target value. Write this function in C/C++.

```
int SequentialSearch(int ar[], int length, int target)
{
//This function searches 0 through length-1 of the incoming array for the target value.
//It returns the number of times the target value is in the array
//If the target is not in the array, the function returns 0
    int lcv, count=0;
    for (lcv=0; lcv<length; lcv++)
        if (ar[lcv]==target) count++;
    return count;
}
```

Notes:

There is no reason to try to end the loop early, since the entire array must be searched to find the number of target elements present.

Remember the comparison statement in C and C++ needs two equal signs, not one.

The Java code is identical.

6.11 Write a Visual Basic function that receives a String array and its length as parameters, finds the largest element in the array and returns its index.

```
Function FindLast(list() As String, length As Integer) As Integer
 'Dim i As Integer
  Dim maxIndex As Integer
  maxIndex=0
  For i=1 To length
    If list(i)>list(maxIndex) Then
      maxIndex=i
    End If
  Next i
  FindLast=maxIndex
End Function
```


Note: The largest element in the array of strings is usually the one falling the farthest to the end of the alphabet. Remember, however, that strings are compared using their ASCII values and all the lowercase letters come after the uppercase letters. All the strings should be in the same case for this code to work correctly.

6.12 Given an array in the order shown below, trace the selection sort algorithm and show what the array would look like after each successive pass.

Selection Sort

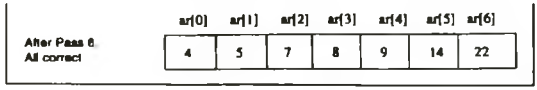
	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]
Original Array	5	14	8	9	22	4	7

	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]
After Pass 1. Last 1 in its spot	5	14	8	9	7	4	22
After Pass 2. Last 2 in correct spots	5	4	8	9	7	14	22
After Pass 3. Last 3 in correct spots	5	4	8	7	9	14	22
After Pass 4. Last 4 in correct spots	5	4	7	8	9	14	22
After Pass 5. Last 5 in correct spots	5	4	7	8	9	14	22
After Pass 6. All correct	4	5	7	8	9	14	22

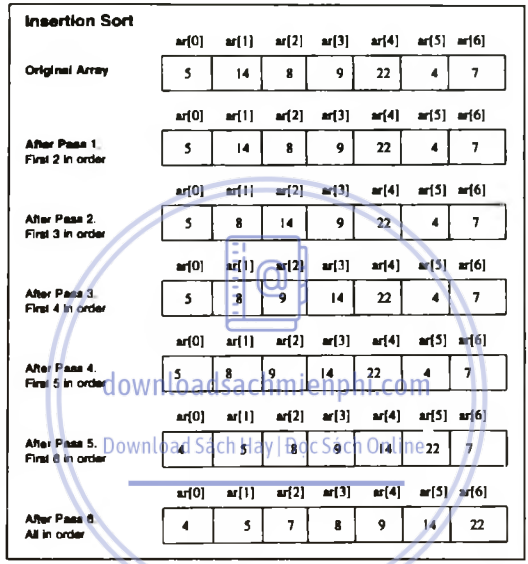
6.13 Given an array in the order shown below, trace the bubble sort algorithm and show what the array would look like after each successive pass.

Bubble Sort

	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]
Original Array	5	14	8	9	22	4	7
After Pass 1. Last 1 in its spot	5	8	9	14	4	7	22
After Pass 2. Last 2 in correct spots	5	8	9	4	7	14	22
After Pass 3. Last 3 in correct spots	5	8	4	7	9	14	22
After Pass 4. Last 4 in correct spots	5	4	7	8	9	14	22
After Pass 5. Last 5 in correct spots	4	5	7	8	9	14	22



6.14 Given an array in the order shown below, trace the insertion sort algorithm and show what the array would look like after each pass.



HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

6.1 Hãy vẽ một hình dùng cho các mảng sau đây:

(a) Một mảng có tên là myScores.

(b) Một mảng có tên là myClass

6.2 Cho một mảng có tên là myArray được khai báo với 10 phần tử, hoặc 10 ô từ [0] cho đến [9], phần nào trong số các phép ?? sau đây sẽ là cuộc gọi hợp lý? Giả sử X=5.

6.3 Vẽ một hình của một mảng có tên là deskItem có chứa một danh sách các hạng mục thường được tìm thấy trên một bàn giấy. Các hạng mục là kẹp giấy, thước nhựa, bút mực và bút chì.

6.4 Vẽ một hình mảng ba chiều có tên là book có chứa 4 trang với

10 dòng, 15 số nguyên trên mỗi trang.

6.5 *Hãy viết mã Visual Basic. C/C++ và Java để khai báo mảng ba chiều trên bài tập trên đây.*

6.6 *Hãy viết các câu lệnh khai báo Visual Basic dành cho cả hai mảng trong bài tập 6.1 trên đây.*

6.7 *Hãy viết câu lệnh khai báo C/C++ dành cho cả hai mảng trong bài tập có lời giải 6.1 trên đây.*

6.8 *Hãy viết các lệnh khai báo Java trong cả hai mảng trong bài tập có lời giải 6.1 trên đây.*

6.9 *Hãy viết các câu lệnh khai báo Visual Basic, C/C++ và Java dùng cho mảng chuỗi trong bài tập có lời giải 6.3. ứng với C++ và Java hãy gán các giá trị lúc bộ nhớ được cấp phát.*

6.10 *Hãy viết một hàm tìm kiếm tuần tự qua đó trả về số lần mà hạng mục có mặt trong mảng, chứ không trả về trị số của giá trị đích. Hãy viết hàm này trong C/C++.*

6.11 *Hãy viết một hàm Visual Basic để nhận một mảng String và chiều của nó làm các tham số, hãy tìm phần tử lớn nhất trong mảng và trả về trị số của nó.*

6.12 *Cho một mảng theo thứ tự được minh họa dưới đây hãy vẽ sơ đồ một thuật toán phân loại chọn lựa và biểu thị diện mạo của mảng sau khi thực hiện mỗi bước thành công.*

6.13 *Cho một mảng theo thứ tự được minh họa dưới đây hãy vẽ đồ thị thuật toán phân loại bọt và biểu thị diện mạo của mảng sau mỗi bước thành công.*

6.14 *Cho một mảng theo thứ tự được minh họa dưới đây hãy vẽ đồ thị thuật toán phân loại chèn và trình bày diện mạo của mảng sau mỗi bước thành công.*

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

6.15 Draw a picture of an array called *cashOnHand* that will keep track of the amount of cash at the end of each month in the year.

6.16 Draw a picture of an array called *listOfNames* that will contain a list of people's names. The names are: W C. Fields, Cary Grant, and Jane Fonda.

6.17 Write the Visual Basic declaration statements:

(a) For an array with subscripts from 1990 to 1999 that will contain the name of the Employee Of The Year for each year

(b) For an array to keep the sales data at the end of each quarter for 10 parts with part numbers starting at 1101. The parts are numbered consecutively.

6.18 Write the C/C++ and Java declaration statements for the array described in Supplementary Problem 6.15 above.

6.19 Write the C/C++ and Java declaration statements for the array described in Supplementary Problem 6.16 above.

6.20 Write the more efficient sequential search algorithm from Fig. 6-19. in Java.

6.21 Write a binary search function for a string array in Visual Basic.

6.22 Write a C/C++ function that receives an integer array and its length as parameters, finds the smallest element in the array and returns its index.

6.23 Given--an array in this initial order, trace the selection sort algorithm and show what the array would look like after each successive pass.

ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]
cat	rat	dog	pig	cow	pup	bat

6.24 Using the array from Supplementary Problem 6.23 (in the same initial order), trace the bubble sort algorithm and show what the array would look like after each successive pass.

6.25 Using the array from Supplementary Problem 6.23 (in the same initial order), trace the insertion sort algorithm and show what the array would look like after each successive pass.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

6.15 Hãy vẽ một hình ảnh của bất kỳ mảng nào có tên là cashOnHand để theo dõi lượng tiền mặt vào cuối mỗi tháng trong năm.

6.16 Hãy vẽ một hình ảnh của một mảng có tên là listOfNames có chứa một danh sách tên người. Các tên là: W.C.Fields Cary Grant, và Jane Fonda.

6.17 Hãy viết các câu lệnh khai báo của Visual Basic.

6.18 Hãy viết các câu lệnh khai báo khi C/C++ và Java dùng cho mảng được mô tả trong bài tập bổ sung 6.15 trên đây.

6.19 Hãy viết các câu lệnh khai báo khi C/C++ và Java dùng cho mảng được mô tả trong bài tập bổ sung 6.16 trên đây.

6.20 Hãy viết thuật toán tìm kiếm tuần tự hiệu quả hơn từ hình 6.19 trong Java.

6.21 Hãy viết một hàm tìm kiếm nhị phân dành cho một mảng chuỗi trong Visual Basic.

6.22 Hãy viết một hàm C/C++ để nhận một mảng nguyên và chiều dài của nó làm các tham số. Hãy tìm phần tử nhỏ nhất trong mảng và trả về trị số của nó.

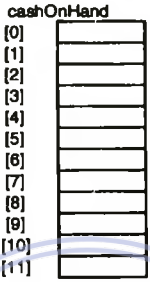
6.23 Cho một mảng theo thứ tự ban đầu, hãy vẽ biểu đồ thuật toán phân loại chọn và trình bày diện mạo của mảng sau mỗi bước thành công.

6.24 Sử dụng mảng từ bài tập bổ sung 6.23 (cũng theo thứ tự ban đầu) hãy thành lập biểu đồ của thuật toán phân loại bọt và biểu thị diện mạo của mảng sau mỗi bước thành công.

6.25 Sử dụng mảng từ bài tập bổ sung 6.23 (theo cùng thứ tự ban đầu giống nhau), hãy thành lập biểu đồ phép phân loại chèn và biểu thị diện mạo của mảng sau mỗi bước thành công.

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

6.15 Array:



6.16 Array:

listOfNames

	[0]	[1]	[2]	[3]	4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
[0]	W	.		C			F	l	e	l	d	a	end string
[1]	C	a	r	y		G	r	a	n	t	end string		
[2]	J	a	n	e		F	o	n	d	a	end string		

6.17 *(a)* `Dim empOfYear (1990 to 1999) As String`

(b) `Dim partsales (1 to 4, 1101 to 1110-) As Currency`

6.18 *(a)* C/C++: `int cashOnHand [ 12 ] ;`

(b) **Java:** `int cashonHand [ ] =new int [ 12 ] ;`

6.19 (a) C/C++:
```
char *listOfNames[3] =("W. C. Fields", "Cary Grant", "Jane Fonda");
```

(b) **Java:**
```
String listOfNames [ ] =new String [3 ] ;
listOfNames[0]="W. C. Fields";
listOfNames[1]="Cary Grant";
listOfNames[2]="Jane Fonda";
```

6.20 The Java code for sequential search:

```
int SequentialSearch (int ar[], int length, int target)
{
//This function searches 0 through length-1 of the incoming array for the target value.
//It returns the index of the target value if it is in the array
//If the target is not in the array, the function returns -1
    int lcv, targetIndex=-1;
    boolean found=false; // boolean type is available in Java
    lcv=0;
    while (lcv<length && !found )
    {
```

```
        if (ar[lcv]==target)
        {
            found=true;
            targetIndex=lcv;
        }
        lcv++;
    }
    return targetIndex;
}
```

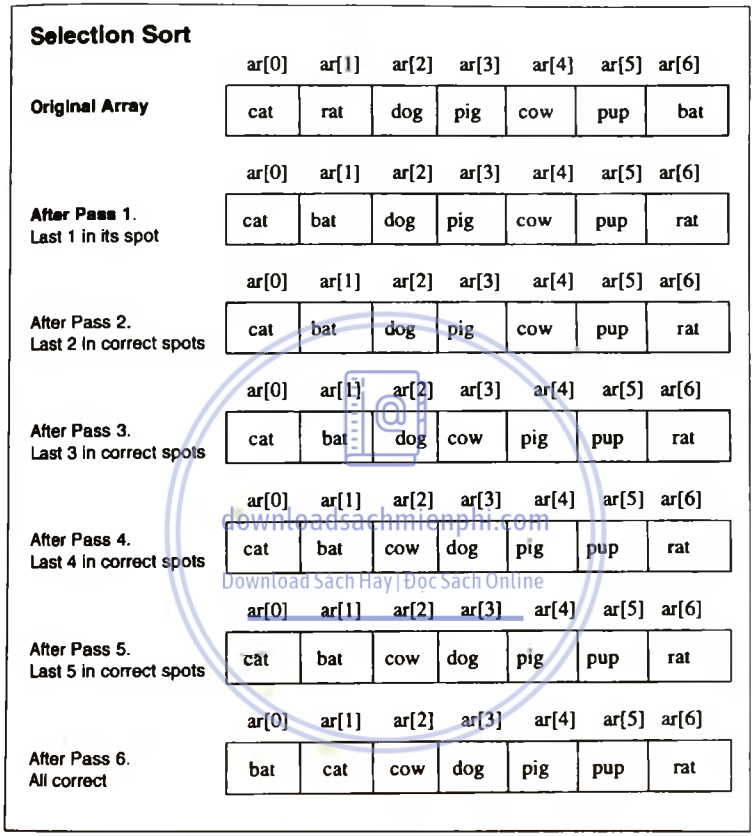
6.21 The Visual Basic code for binary search:

```
Function BinarySearch(ar() As String, length As Integer, _
                      target As String) As Integer
'This function searches 1 through length of the incoming array for the target value.
'It returns the index of the element holding that value.
'If the target is not in the array, the function returns -1
  Dim mid As Integer, lowerBound As Integer, upperBound As Integer
  Dim targetIndex As Integer
  Dim found As Boolean
  'set initial values
  targetIndex=-1
  found=False
  lowerBound=0
  upperBound=length-1
  Do While (lowerBound<=upperBound And Not found)
    mid=(upperBound+lowerBound) \ 2
    If (target<ar(mid)) Then
      upperBound=mid-1 'eliminate the upper half of this section
    ElseIf (target>ar(mid)) Then
      lowerBound=mid+1 'eliminate the lower half of this section
    Else
      targetIndex=mid 'target is found
      found=True
    End If
  Loop
  BinarySearch=targetIndex
End Function
```

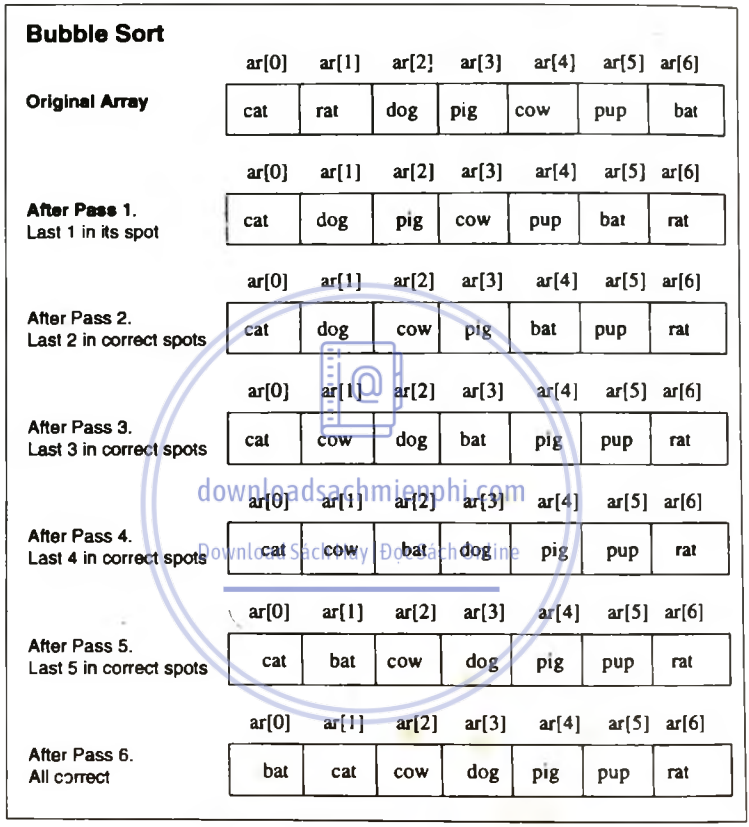
6.22 The C/C++ code for finding the minimum value in an integer array

```
int FindMin(int ar[], int length)
{
//This function searches 0 through length-1 of the incoming array for the
//lowest value.
//It returns the index of the lowest value
    int lcv, lowest=0;
    for (lcv=1; lcv<length; lcv++)
        if (ar[lcv]<ar[lowest]) lowest=lcv;
    return lowest;
}
```

.23 The selection sort trace:



6.24 The bubble sort trace:



6.25 The insertion sort trace:

Insertion Sort

	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]
Original Array	cat	rat	dog	pig	cow	pup	bat
After Pass 1. First 2 in order	cat	rat	dog	pig	cow	pup	bat
After Pass 2. First 3 in order	cat	dog	rat	pig	cow	pup	bat
After Pass 3. First 4 in order	cat	dog	pig	rat	cow	pup	bat
After Pass 4. First 5 in order	cat	cow	dog	pig	rat	pup	bat
After Pass 5. First 6 in order	cat	cow	dog	pig	pup	rat	bat
After Pass 6. All in order	bat	cat	cow	dog	pig	pup	rat

HƯỚNG DẪN ĐỌC HIỂU TRẢ LỜI CHO BÀI TẬP BỔ SUNG

6.15 Mảng:

6.16 Mảng:

6.20 Mã Java dành cho phép tìm kiếm tuần tự là:

6.21 Mã Visual Basic dành cho phép tìm kiếm nhị phân nào.

6.22 Mã C/C++ dành để tìm giá trị cực tiểu trong một mảng nguyên là:

6.23 Đồ thị của phép sắp xếp theo phương pháp chọn trực tiếp

6.24 Đồ thị của phép sắp xếp theo phương pháp nổi bọt

6.25 Đồ thị của phép sắp xếp theo phương pháp chọn trực tiếp

CHAPTER 7

Data files

Các file dữ liệu

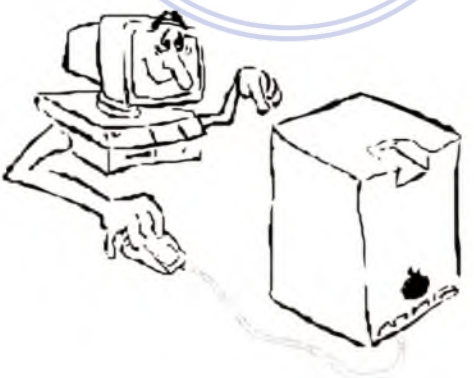
MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các khái niệm về file, hệ thống file, cách sắp xếp và quản lý các file dữ liệu, ... , cụ thể với các nội dung cơ bản sau đây:

- Introduction
- Data terminology
- File organization
- Text and binary files
- Opening and closing files

- *Giới thiệu*
- *Thuật ngữ dữ liệu*
- *Cách tổ chức file*
- *Các file văn bản và nhị phân*
- *Mở và đóng các file*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.



CHỦ ĐIỂM 7.1

INTRODUCTION
Giới thiệu

The file system is an essential component of any computer. Most of the applications and their data reside in files stored in a hard disk, diskette, or some other appropriate storage medium. In computer terminology. anything that can be stored in a diskette or hard disk is a file. Files are given unique names and may contain data of any type. In general, files are named using the following convention:

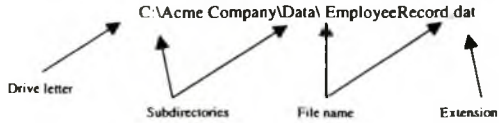
[pathname]filename[.extension]

where the square brackets indicate that the filename may include an optional pathname, and an optional extension. For most personal computers, the **pathname** starts with a single drive letter and a directory. The drive letter, followed by a colon, indicates the disk drive where the file resides. Following the drive letter, pathnames may include a directory and some subdirectories that explicitly state the location of the file in the computer system. Directories start at the root directory and, depending on the system, each subdirectory may be separated from the previous one by either a backslash or frontslash. The filename is a sequence of characters consisting of letters, numbers, and some punctuation symbols such as a blank, hyphen, or an underscore. In some computer systems, the name given to a file cannot exceed more than eight characters. In some other systems, only letters and numbers can be part of a file name. File names should be chosen to indicate both the type of data contained in the file and the application for which it is being used. Following the filename there may be a file extension. This extension, separated from the file name by a period, is generally an optional three-character sequence used to describe the content of the file. This file name convention is sometimes referred to as the first name, middle name, and last name of the file. In this case, the first name is the file name, the middle initial is the period and the last name is the extension. Files names are unique within a particular folder or subdirectory. In other words, no system will allow two files with identical first name, middle initial, and last name in the same folder or subdirectory.

EXAMPLE 7.1 Identify the elements of the following filenames:

(a) C:\Acme Company\Data\Employee Reccord.dat

(b) D:\My Documents\tax records.dat



The components of tile name described in (a) are:

The elements of the file name described in *(b)* are:

EXAMPLE 7.2 Determine whether or not the following sequences are valid file names:

(a) TaxDocuments.doc *(b)* AveryLongNameForADataFile.exe (c) ?>.doc

Names (a) and *(b)* are valid for most computer systems or applications. However, there may be applications where these names are illegal because they are longer than eight characters. Name (c) is illegal because it contains invalid characters such as the ? or the >.

Depending upon the convention being used, the extension of a file may determine its type. Example 7.3 clarifies this.

EXAMPLE 7.3 Classify the files shown below according to their extension.

(a) BirthDate.doc *(b)* List.txt *(c)* Employee.mdb *(d)* tax.dat

The extension .doc is generally used to identify documents created with word processors such as MS Word.

The extension .txt is generally used to identify files created with text editors such as MS Notepad. Files with a .txt extension are generally referred to as "plain text" or ASCII files.

The extension .mdb is generally used to identify database files created using a database management system such as MS Access.

The extension .dat of problem *(d)* is generally used to store "data" of some sort.

Files do not need to have an extension; however, some applications require that a particular extension be used if we want to launch the application in some systems by double clicking on a file. For example, if we want to launch the MS C++ compiler automatically by double clicking on a source file, the file must have a .c or .cpp extension.

CHÚ THÍCH TỪ VỰNG

pathname:	*tên đường dẫn*
filename:	*tên file*
file extension:	*phần mở rộng*
record:	*bảng ghi*
field:	*trường*
key field:	*trường khóa*
sequential:	*tuần tự*
random:	*ngẫu nhiên*
read:	*đọc*
text file:	*file dạng text*
binary files:	*file dạng nhị phân*
opened:	mở
closed:	đóng
modes:	phương thức
output:	đầu ra

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 7.1

Hệ thống file là thành phần cơ bản của bất cứ máy tính nào. Hầu hết các trình ứng dụng và dữ liệu của nó đều nằm trong các file lưu trữ trong đĩa cứng, đĩa mềm hoặc một vài phương tiện lưu trữ phù hợp khác, theo thuật ngữ máy tính thì bất cứ nội dung nào có thể được lưu trữ trong một đĩa mềm hay một đĩa cứng đều là một file. Các file đều được cung cấp một tên duy nhất và có thể chứa dữ liệu bất cứ kiểu nào. nói chung các file được đặt tên bằng cách sử dụng quy ước sau đây:

[pathname]filename[.extension]

[tên đường dẫn, tên file, phần mở rộng]

trong đó các dấu móc vuông chỉ ra rằng tên file có thể chứa một đường dẫn tùy ý và một phần mở rộng tùy ý đối với hầu hết các máy tính cá nhân, thì đường dẫn bắt đầu bằng một mẫu tự ổ đĩa và một thư mục. Mẫu tự ổ đĩa, theo sau là một dấu (:), chỉ ra ổ đĩa nơi mà file đang có. Theo sau mẫu tự ổ đĩa, tên của đường dẫn có thể chứa một thư mục và một vài thư mục con để phát

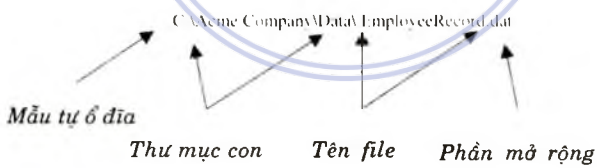
biểu một cách minh nhiên vị trí của file trong hệ thống máy tính. Các thư mục bắt đầu tải thư mục gốc và phụ thuộc vào hệ thống, mỗi thư mục con có thể được rời với thư mục con trước đó bằng một dấu backslash hoặc dấufronslash. Các tên file là một chuỗi các ký tự có chứa các mẫu tự, các số và một vài ký hiệu phép chấm câu, chẳng hạn như một khoảng trống, một dấu gạch nối hoặc một dấu gạch dưới. Trong một vài hệ thống máy tính, tên được cung cấp cho một file không được vượt quá 8 mẫu tự. Trong một vài hệ thống khác, chỉ có các mẫu tự và những con số có thể là một phần của tên file. Các tên file phải được chọn để chỉ rõ kiểu dữ liệu được chứa trong file và ứng dụng mà file đang được dùng. Theo sau tên file là có thể có một phần mở rộng file. Phần mở rộng này tách rời tên file bằng một dấu chấm, thường là một chuỗi 3 ký tự được dùng để mô tả nội dung của file. Quy ước đặt tên file đôi khi tham chiếu làm tên chữ lót và họ của file. Trong trường hợp này thì tên chính là tên file, chữ lót ở giữa là dấu chấm và họ chính là phần mở rộng. Các tên file là duy nhất bên trong một folder đặc biệt hoặc một thư mục con. Nói cách khác không có hệ thống nào cho phép có hai file có tên giống nhau, chữ lót giống nhau và họ giống nhau ở trong cùng một thư mục hoặc một thư mục con.

VÍ DỤ 7.1 *Hãy chỉ định các thành phần của tên file sau đây:*

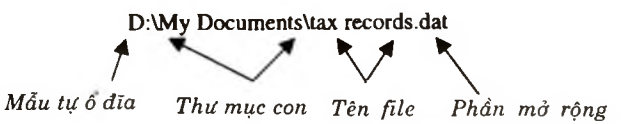
(a) C:\Acme Company\Data\Emploce Reccord.dat

(b) D:\My Documents\tax records.dat

Các thành phần của tên file được mô tả trong (a) là:



Các thành phần của tên file được mô tả trong (b) là



VÍ DỤ 7.2 *Hãy xác định cho biết chuỗi sau đây có phải là các tên file đúng hay không?*

(a) *TaxDocuments.doc*

(b) *AveryLongNameForAdataFile.exe*

(c) *?>.doc*

Các tên (a) và (b) là đúng ứng với hầu hết các hệ thống máy tính hoặc các trình ứng dụng. Tuy nhiên, có thể có các trình ứng dụng ở đó có tên file này không hợp lý bởi vì chúng dài hơn 8 mẫu tự, tên (c) là không hợp lý bởi vì nó có chứa các ký tự không cho phép, chẳng hạn như dấu (?) hoặc dấu (>).

Phụ thuộc vào quy ước được dùng, phần mở rộng của một file có thể xác định kiểu của nó. Ví dụ 7.3 làm rõ điều này.

***VÍ DỤ 7.3** Hãy làm rõ các file được minh họa dưới đây theo phần mở rộng của chúng.*

(a) *BirthDate.doc*

(b) *List.txt*

(c) *Employee.mdb*

(d) *tax.dat*

Phần mở rộng .doc thường được dùng để chỉ định các tài liệu được tạo ra với chương trình xử lý văn bản chẳng hạn như MS Word.

Phần mở rộng .txt thường được dùng để chỉ định các file được tạo ra với các trình biên soạn text chẳng hạn như MS NotePad. Các file có phần mở rộng .txt thường được tham chiếu làm các file "thuần túy văn bản" hoặc các file ASCII.

Phần mở rộng .mdb thường được dùng để chỉ định các file cơ sở dữ liệu được tạo ra bằng cách dùng một hệ thống quản lý cơ sở dữ liệu chẳng hạn như MS Access.

Phần mở rộng .dat của bài tập (d) thường được dùng để lưu trữ "dữ liệu" của một vài loại.

Các file không cần phải có một phần mở rộng; tuy nhiên, một vài trình ứng dụng yêu cầu rằng phải có một phần mở rộng đặc biệt được dùng nếu bạn muốn khởi động một trình ứng dụng đó trong một vài hệ thống bằng cách nhấp đúp lên một tên file. Ví dụ nếu bạn muốn khởi động chương trình biên soạn MS C++ tự động bằng cách nhấp lên trên tên file nguồn, thì file đó phải có phần mở rộng là một .c hoặc .cpp.

CHỦ ĐIỂM 7.2

DATA TERMINOLOGY
Thuật ngữ dữ liệu

Files are made up of **records.** Generally, there is one type of record per file. Records are made up of fields. For example, consider the file made up of employee records as shown in Table 7-1. There is one record per employee. Each row of this table constitutes a record.

Table 7-1

Id	Last Name	First Name	Department	Salary	Email
0155	Johnson	Michael	Accounting	45,000	johnmic@ccnet.com
0352	Heffner	Lois	Marketing	38,000	heffloi@ccnet.com
9461	Tsu	Tan	M.I.S.	35,000	tsutan@ccnet.com
9830	Hudson	Michael	Accounting	45,000	hudsmic@ccnet.com

The fields of these records are Id, Last Name, First Name, Department, Salary and Email. All records share the same structure. That is, every record has the same fields and every field appears, within the record, in the same order. Records in data files are generally organized according to a particular field called the key. The **key field** serves a dual purpose. First, it is used uniquely to identify each record. Secondly, the key field is used to sort the records, according to their values in this field, in ascending or descending order. For instance, Id is the key field in Table 7-1. Notice that the records, in this case, are sorted on the Id field in ascending numerical order. A key field may be numerical or it may be a string of characters.

Files have special markers to indicate where they begin and where they end. These two markers are generally called BOF (Beginning Of File) and EOF (End Of File) respectively. When a file is empty, that is, when it does not have any records in it, both markers coincide.

The reader needs to be aware that not every programming language uses or even has the notion of a record. For instance, C++ does not impose any structure on files. Therefore, the notion of a "record" does not exist in C++. In C++, a file is nothing more than a sequence of bytes that ends in a special character **called the end-of-file** marker or after a specific number of bytes recorded in an administrative data structure.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 7.2

Các file làm nên các record (bảng ghi). Thông thường có một kiểu record trên mỗi file. Các record lại hình thành nên các filed (trường). Ví dụ khảo sát file hình thành nên các record của nhân viên như minh họa trong bảng 7.1. có một record trên mỗi nhân viên. Mỗi hàng của bảng này có chứa một record.

Bảng 7.1

Id	Last Name	First Name	Department	Salary	Email
0155	Johnson	Michael	Accounting	45,000	johnmic@ccnet.com
0352	Heffner	Lois	Marketing	38,000	heffler@ccnet.com
9461	Tsu	Tan	M.I.S.	35,000	tsutan@ccnet.com
9830	Hudson	Michael	Accounting	45,000	hudsmic@ccnet.com

Các trường của các record này là Id, Last Name, First Name. Department Salary, và Email. Các record này chia sẻ cấu trúc giống nhau. Có nghĩa rằng mỗi một record thì có trường giống nhau và mỗi trường lại xuất hiện bên trong record theo cùng một thứ tự. Các record trong các file dữ liệu thường được tổ chức theo một trường đặc biệt được gọi là trường khóa (trường chính). Các trường khóa phục vụ hai mục đích. Trước tiên nó được dùng một cách duy nhất để nhận biết mỗi record. Thứ hai trường khóa được dùng để phân loại các record theo giá trị của nó trong trường theo thứ tự tăng dần hoặc giảm dần. Ví dụ Id là trường khóa trong bảng 7.1 lưu ý rằng các record trong trường hợp này được phân loại tên trường Id theo thứ tự số tăng dần. Một trường khóa có thể là một số hoặc nó có thể là một chuỗi các ký tự.

Các file thì có các dấu kiểm đặc biệt để chỉ định nơi mà chúng bắt đầu và nơi mà chúng kết thúc. Hai dấu kiểm thường có tên là ***BOF*** *(Beginning Of Fie) và* ***EOF*** *(**End Of File**) tương ứng. Lúc một file là trống có nghĩa là lúc nó không có bất cứ record nào trong đó thì cả hai dấu kiểm này trùng nhau.*

Người đọc cần phải cảnh báo điều này là không phải mỗi một ngôn ngữ lập trình đều sử dụng hoặc thậm chí có một chú thích record. Ví dụ C++ thì không có bất cứ cấu trúc nào về các file do đó phần chú thích về một record không hiện diện trong C++. Trong C++ một file không gì khác hơn là một chuỗi các file vốn kết thúc theo một ký tự đặc biệt được gọi là dấu kiểm end-of-file hoặc sau một số các file đặc biệt được ghi chú trong một cấu trúc dữ liệu quản lý.

CHỦ ĐIỂM 7.3

FILE ORGANIZATION
Cách tổ chức file

This term refers to the way data are organized, stored, and retrieved. There are basically two types of file organization: **sequential and random.** A sequential organization indicates that the records are stored one after another in some sequence. When these records are retrieved or read they must be retrieved in the same order in which they were stored. This implies that in order to read the fifth record of a sequential file we must first retrieve the previous four records. In a random file we can read the records in any order. Although this offers more flexibility than a sequential organization, it requires that all records be of the same length. In a sequential file the individual records may be of different lengths.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 7.3

Thuật ngữ này ám chỉ đến cách mà dữ liệu được tổ chức, được lưu trữ và truy xuất. Có hai kiểu tổ chức file căn bản là tổ chức tuần tự và ngẫu nhiên. Tổ chức tuần tự chỉ ra rằng các record được lưu trữ lần lượt theo một trình tự. Lúc record này được truy xuất được đọc, chúng cũng được truy xuất theo từng thứ tự mà chúng được lưu trữ. Điều này ngụ ý rằng để đọc record thứ 5 của một file tuần tự trước khi bạn phải truy xuất 4 record trước đó. trong một file ngẫu nhiên chúng ta có thể đọc các record bất cứ thứ tự nào mặc dù điều này thường linh động hơn là tổ chức tuần tự, nhưng nó yêu cầu rằng tất cả record phải có chiều dài giống nhau. Trong một file tuần tự các record riêng biệt có thể có chiều dài khác nhau.

CHỦ ĐIỂM 7.4

TEXT AND BINARY FILES
Các file văn bản và nhị phân

In addition to their organization, files can be classified according to the way their constituent bytes are interpreted by the computer system. There are two basic modes: text and binary. In text files, each byte is interpreted as an ASCII character. In **binary tiles,** the bytes are not interpreted in any manner. In this case it is necessary to know beforehand the content of the file. That is, we must know what was stored in the file in the first place. Binary files cannot be read by word processors or text editors. Text files, on the contrary, can be read and created with any text editor. A text file can also be created with a word processor provided that the file is saved as "text" or ASCII.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 7.4

Bên cạnh tổ chức, các file có thể được phân loại theo cách mà các file liên tục của chúng được diễn dẫn bởi một hệ thống máy tính. Có hai chế độ căn bản đó là chế độ text và chế độ nhị phân. Trong các file text mỗi một file được diễn dịch với một ký tự ASCII. Trong các file nhị phân các byte không được diễn dịch theo bất cứ cách thức nào. trong trường hợp này chúng ta cần phải biết trước khi xử lý nội dung của file. Có nghĩa là chúng ta phải biết những gì được lưu trữ trong file ở vị trí đầu tiên. Các file nhị phân không thể được đọc bởi các chương trình xử lý văn bản hoặc các chương trình biên soạn text. Ngược lại các file text có thể được đọc và được tạo ra với bất kỳ chương trình biên soạn text nào. một file text cũng có thể được tạo ra với một chương trình xử lý văn bản do file đó cung cấp và được lưu dưới tên "text" hoặc ASCII.

CHỦ ĐIỂM 7.5

OPENING AND CLOSING FILES
Mở và đóng các file

There is no standard way to operate on files across the different programming languages. In general, each language has its own peculiar way to handle files. Moreover, programming languages are not even required to provide any I/O mechanisms. For example, C does not have built-in capabilities for performing I/O. It is up to the manufacturers of the C compilers to provide all the necessary I/O capabilities. Recall from Chapter 3 that Java programs can be either applets for use on Web pages or stand-alone applications. For security reasons, Java applets cannot read from or write to local drives. Java applications, however, make extensive use of class objects for file input and output. Classes will be explained in Chapter 8, but the complexity of Java file input and output classes is beyond the scope of this book.

Regardless of the language, there are some "common file operations" that need to be performed when working with files. Before using a file, and depending on the language, it is necessary to associate with the file a unique numerical identifier or a pointer by which the file will be referred to later during the execution of the program. This unique identifier is sometimes called a "file handler." The association between a file and its handler usually occurs when the file is opened. This association ceases to exist when, at the end of the program, the file is closed.

7.5.1 Opening Files in Visual Basic - *Mở các tập tin trong Visual Basic*

The format of the instruction that allows us to open a file in Visual Basic is as follows:

Open "filename" For (Input I Output I Append I Random) As " Wenumber" [Len = RecLength]

where filename must be enclosed in double quotes and may include the entire path to the file. The curly braces in the instruction indicate that we have to choose one and only one of the options enclosed by the braces; the vertical bars separate the different options from which we can choose. That is, we need to open the file ***in one and only one*** of the four modes as indicated in the instruction. Table 7-2 explains the purpose of the different modes. The filenumber can be any integer from 1 to 511. The Len = RecLength -option applies to random files only. A record can be up to 32,767 characters long.

Table 7-2

Mode	Type of File	Action Required to
Input	Sequential	read records from a file.
Output	Sequential	write records to a file starting at the BOF. Previous data are overwritten.
Append	Sequential	write records to the end of the file (append).
Random	Random	read/write records in any order.

The filenumber is uniquely associated with a file as long as the file is opened. Once the file is closed, this number can be assigned to another file that needs to be opened.

EXAMPLE 7.4 Write the necessary Visual Basic instructions to open the file InFile.dat for input and the file OutFile.dat for output. Assume that both files are located in a subdirectory called C:\vb\project.

```
Open "C:\vb\project\InFile.dat" for Input as #1
Open "C:\vb\project\OutFile.dat" for Output as #2
```

The input and output file have been assigned the numbers #1 and #2 respectively; however, if both files need to be opened simultaneously in the same program, any two different integer numbers between 1 and 511 could have been assigned to these files.

7.5.1.1 The FreeFile Function - Hàm FreeFile

As previously indicated, to open a file a user can assign to the file any integer number between 1 and 511. This is an easy task when there are few files to manipulate; however, when there are a large number of files, it is better to use the FreeFile function to assign the next available number to a file. This way the user avoids the inconveniences of assigning the same number to more than one file at the same time. The use of the FreeFile function is illustrated in the next example.

EXAMPLE 7.5 Rewrite the instructions of the previous example using the FreeFile function.

```
Dim iInFile As Integer 'This variable is used to open the
  input file
Dim iOutFile As Integer 'This variable is used to open the
  output file
iInFile=FreeFile
Open "C:\vb\project\InFile.dat" for Input as #iInFile
  iOutFile=FreeFile
Open "C:\vb\project\OutFile.dat" for Output as #iOutFile
```

The FreeFile function is called *before* each open instruction to obtain

the next available number. In each case, the value returned by the FreeFlle function is assigned to an integer variable that is then used in the open statement.

7.5.1.2 The Close Statement - Câu lệnh Close

When a program has finished processing a file, the file needs to be closed. There are two instructions that can be used to do this.

Close #filenumber closes the file to which it was assigned the integer filenumber. All other files, if any, remain opened.

Close closes all opened files. Notice that in this case we do not have to specify any file number.

7.5.1.3 Reading Data From Sequential Files - Đọc dữ liệu từ các file liên tục

As indicated before, the data in any sequential file are read in the same order in which they were written. As Example 7.6 shows, when a sequential file is open, the user starts reading from the beginning of the file. The instruction that allows us to read records from a sequential file has the following format:

Input #Filenumber, field1, field2, ...,fieldn

where **Input** is a keyword that indicates that we are reading (inputting) data from a file which was opened in Input mode. Filenumber is the integer number assigned to the previously opened file from which we are reading.

When preparing a sequential file for input, character strings are enclosed in double quotes and separated by commas. Sequential files are usually created with an editor or with a word processor. However, when created with a word processor, sequential files need to be saved as text.

EXAMPLE 7.6 Assume that a sequential file (models.dat) contains the different car models that Silverado Motors has currently on stock. Write a Visual Basic procedure that reads this sequential file and displays its content in a listBox called lStMOdelSOnStock. Use the test file shown below.

Input Text file:

"Xavier LTD", "Cougar 100", "El Dorado Special", "Silver Bullet 2000",

"Bronco all terrain", "Ford Globe Trotter", "Audi 2000", "Chevrolet El Camino"

The code of the procedure may look like this:

```
Private Sub ReadModels()
Dim stModels As String 'This variable is used to read the
  models in
Dim iLoopIndex As Integer 'This variable is used as index
```

```
    of the for loop
  Open "C:\temp\Vbjunk\models.dat" For Input As #1
    For iLoopIndex=0 To 7
     Input #1, stModels
    DisplayForm.lstK\odelsOnStock.AddItem stModels,
    iLoopIndex
  Next iLoopIndex
```

In this example, we have assumed that the listBox appears on a form called DisplayForm. Notice that the .AddItem method has been used to fill in the listBox. The variable iLoopIndex serves a dual purpose. First, it is used to control the execution of the loop. Secondly, since the loop index starts at zero it serves also as an index to the list. A list index in Visual Basic begins at zero. Since there are eight models in the input file, the iLoopIndex variable varies from 0 to 7. The general format of the instruction to fill the listBox is as follows:

```
Object.Additem value [,index]
```

The square brackets indicate that we have the option of using an index.

The code of Example 7.6 assumes that there are exactly eight different models in the input file. Solved Problem 7.2 shows a more general method to read data from a file without requiring to know beforehand the number of records in the file.

7.5.1.4 Writing Data To a Sequential File **- Ghi dữ liệu vào một file liên tục**

To write data to a sequential file use the Write instruction. The general format of this instruction is:

Write #pilenumber, fieldl, field2, field3, ...fieldn

where #Filenumber is a file already opened in Output or Append mode and fieldl, field2,. . .fieldn are the different fields that we want to write to that file. We refer collectively to all these fields as the "list of fields." When Visual Basic writes these fields to disk, it separates them with commas. String data are enclosed in double quotes and numerical data are written "as is." After writing the last element of the list, Visual Basic automatically inserts a carriage return and a line feed into the file.

Table 7-2 shows the two different modes that can be used to write records to a file: Output and Append. When output mode is used, the data are written to the file starting at the beginning of the file. This mode overwrites any data already on the file. When append mode is used, the data are added at the end of the file following the last piece of information previously written to the file.

7.5.1.5 The *End Of File Function* - Hàm End Of File

In Example 7.6, to execute the program correctly we needed to know the exact number of elements that needed to be entered into the list. A more general approach to fill in the list is to read the different car models until we run out elements in the input file. In other words, we read the different car models until we reach the end of the file. The EOF function allows us to detect when we have reached the end of the file. The general format of the EOF function is as follows:

EOF (Filenumber) where Filenumber is the integer associated with a previously opened file in input mode.

The function EOF() returns a True value when the end of file marker has been detected. This occurs after the last data in the file has been read in or as soon as we try to read from a file that is originally empty. Solved Problem 7.2 illustrates the use of the EOF function.

7.5.2 Files in C++ - *Các tập tin trong C++*

C++ views files as a stream of bytes. Although this view is shared by all the compiler manufacturers, as of the writing of this book there is not a definite standard on the specification of the I/O system. In this section, we will discuss the most commonly used 1/O system. This system, called the C++ I/O stream, is based on three different classes: the istream for input, the ostream for output and the iostream for input/output. These classes allow the user to define and manipulate files. The topic of classes will be addressed more fa~rmally in Chapter 8.

Whenever we start a C++ (program, there are four different class variables that are created automatically. These classes are indicated in Table 7-3.

Table 7-3

Variable	Use
cin	Console Input/generally assigned to the keyboard
cout	Console Output/generally assigned to the screen
cerr	Console **Error**/ generally assigned to the screen
clog	Console Log/ generally assigned to the screen

All these class variables are defined in the header file <iostream.h>. It is important to keep in mind that we need to select the appropriate version or mode of the stream class when doing disk I/O. The I/O stream classes, indicated in Table 7-4, are-included in the header file <iostream.h>.

```
inFile.open("c:/temp/data.dat");   //location of the
  input file
outFile.open("c:/temp/data.out");  //location of the
  output file
inFile>>i; //read the upper limit of the loop index for
  (loopIndex=1; loopIndex<=i; loopIndex++)
                outFile<<loopIndex<<endl;
inFile.close();
outFile.close();1
```

Observe that in the specification of the path for both the input and output files we have used the character "/" instead of the usual "\". If we do not this the compiler will generate a warning because, in the family of C-like languages, the character "\" is interpreted as the escape character.

7.5.3 Files in C - *Các tập tin trong C*

To reference a file in C, the user needs to declare a variable of type File. The general format for declaring variables of this type is

FILE * variable_name;

where FILE is a special type of data structure contained in the standard header file *stdio.h* that contains information about the current status of the file. Variable-name refers to the name of the pointer variable selected by the user.

EXAMPLE 7.10 Declare two file variables named InFile and OutFile.

According to format indicated above, these two variables can be declared as follows:

FILE * InFile;

FILE * OutFile;

This example assumes that stdio.h has been included at the beginning of the program.

7.5.3.1 Opening Files in C - Mở các tập tin trong C

In C, the open file instruction serves a dual purpose. First, it establishes a physical connection between the program and the data file. Secondly, it equates the external name of the file with the pointer variable declared in the program.

To open a file in C we use the function fopen(). This function takes, as indicated below, two arguments and returns a pointer to the file if the operation succeeded. If the file cannot be opened, fopen() returns a NULL. A filename may or may not include the entire path to the file.

fopen(I'Mename","access mode");

The access mode indicates the type of operations that the user intends to perform on a file. Table 7-5 shows the complete set of options for accessing a file. In this book we will only consider the r, w, and a options.

EXAMPLE 7.11 Using the declarations of the previous example, open the file InFile for reading and open the file Oufle for writing. Assume that the locations and names of the input and output files are "C:/temp/inData.dat" and "C:/temp/OutData.dat" respectively.

We can open these two files as follows:

```
InFile=fopen("C:/temp/inData.dat","r");
OutFile=fopen("C:/temp/OutData.dat","w");
```

Table 7-5

Mode	Use
r	Opens the file for read operations only
w	Opens the file for writing. If the file already exists, its contents are erased. Otherwise the file is created.
a	Opens the file for appending. If the file does not exist it is created.
r+	If the file already exists it opens it for both reading and writing. Otherwise, it returns an error.
w+	Creates a file and opens it for both reading and writing. If the file already exists, its contents are erased.
a+	Opens the file for reading and appending if the file already exists. Otherwise, it creates a new file.

7.5.3.2 Using Files in Input and Output Instructions - Sử dụng các tập tin trong các lệnh nhập và xuất

Once the files have been opened we can use the functions fscanf(), fgets(), or fgetc() to read from the file. Likewise, we can use the functions fprintf(), fputs(), or fputc() to write to the file. In the remainder of this chapter we will use the functions fscanf and fprintf to do I/O operations since they are more flexible than fgets, fgetc, fputs, or fputc. The general format of fscanf and fprintf is as follows:

fscanf(pointer to_61e,format string, &arguments)

fprintf(pointer_to_file,formatstring, arguments)

EXAMPLE 7.12 Assume that you have an input file that contains the data shown below. Write the necessary C instructions to read four records

from the input file and write these records to another output file according to the given formats.

Input file (indata.dat)	Output File (outdata.dat)
Smith Joe	Joe Smith
Randall George	George Randall
Weaver Earl	Earl Weaver
Ford James	James Ford

In this program, we have assumed that both the first and last names are limited to a maximum of 20 characters each. We have also used the function fscanf() to read the strings from the input file and the function fprintf() to write the strings to the output file. Notice also that to specify the path of both the input and output files we have used the frontward slash "/" instead of the backward slash "\". In, addition, we have made provision to exit the program if there is an error when opening the file.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{ int loopIndex; char Fname[20], Lname[20]; //arrays to
  hold first and last names
FILE *inFile;
FILE *outFile;
outFile=fopen("c:/temp/outdata.out","w");//open file
  for output
/*open file for input. If there is an error, exit program*/
if ((inFile=fopen("c:/temp/indata.dat","r"))==NULL)
{ printf("\nFailed to open input file");
  exit (1); }
/*Read strings from the input file and write them to the
  output file*/
for(loopIndex=1; loopIndex<=4; loopIndex++)
{ fscanf(inFile,"%s %s",Fname,Lname);
  fprintf(outFile,"%s %s\n",Lname,Fname); }
} //end of program
```

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 7.5

Không có cách thức chuẩn mực nào để thao tác trên các file ngang qua nhiều ngôn ngữ lập trình khác nhau. Nói chung thì mỗi ngôn ngữ có một cách thức riêng biệt của nó để xử lý các file. Ngoài ra các ngôn ngữ lập trình thậm chí không yêu cầu phải cung cấp bất kỳ cơ cấu nhập xuất nào. Ví dụ C không có các khả năng được tạo sẵn để thực hiện I/O. Tính ra thì các nhà sản xuất các chương trình C phải cung cấp tất cả khả năng I/O cần thiết. Hãy nhớ lại từ chương 3 rằng các chương trình Java có thể hoặc là các chương trình ứng dụng nhỏ để sử dụng trên các trang web hoặc các chương trình ứng dụng đứng riêng. Vì lý do an toàn, các trình ứng dụng nhỏ của Java không thể đọc hoặc viết sang các ổ đĩa cục bộ. Tuy nhiên, các chương trình ứng dụng Java làm cho việc sử dụng các đối tượng lớp được mở rộng dành cho việc nhập và xuất file. Các lớp sẽ được giải thích trong chương 8, nhưng tính phức tạp của các lớp nhập và xuất của file Java vượt xa phạm vi thảo luận trong sách này.

Bất kể ngôn ngữ lập trình nào, vẫn có một vài "thao tác file chung" cần phải được thực hiện lúc làm việc với các file. Trước khi sử dụng một file vào thủ tục và ngôn ngữ bạn cần phải liên kết với một file với một bộ định dạng số duy nhất hoặc một con trỏ qua đó file sẽ được tham chiếu sau này trong suốt quy trình thực thi chương trình. Bộ định dạng đồng nhất này đôi khi còn được gọi là "bộ xử lý file". Sự liên kết giữa một file và bộ xử lý của nó thường xảy ra lúc file được mở. Liên kết này chấm dứt sự hiện hữu lúc ở cuối một chương trình, file được đóng.

7.5.1 Các file mở trong Visual Basic

Dạng câu lệnh cho phép chúng ta mở một file trong Visual Basic như sau:

Open "filename" For {Input/Output/Append/random} As "#filenumber" [Len=RecLength]

Ở đây tên file phải được đặt trong các dấu ngoặc kép và có thể đưa vào toàn bộ đường dẫn đến file. Các dấu móc vuông trong câu lệnh chỉ ra rằng chúng ta phải chọn một và chỉ một trong số các tùy chọn được đặt bên trong các dấu móc. Hoặc thẳng đứng tách rời các tùy chọn khác so với tùy chọn mà chúng ta có thể chọn. Có nghĩa rằng chúng ta phải mở file theo một và chỉ một trong số 4 chế độ như được chỉ định trong câu lệnh. Bảng 7.2 giải thích mục đích của các chế độ khác nhau. Số file (filenumber) có thể là một số nguyên bất kỳ từ 1 cho đến 511. Tùy chọn Len = RecLength là chỉ áp dụng cho các file ngẫu nhiên. Một record có thể chứa 32.767 ký tự.

Bảng 7.2

Mode	Type of File	Action Required to
Input	Sequential	read records from a file.
Output	Sequential	write records to a file starting at the BOF. Previous data are overwritten.
Append	Sequential	write records to the end of the file (append).
Random	Random	read/write records in any order.

Số file là duy nhất và được liên kết với một file khi file được mở. Một khi file đã đóng thì con số này có thể được gán cho một file khác cần phải được mở.

***VÍ DỤ 7.4** Hãy viết các câu lệnh Visual Basic cần thiết để mở file **InFile.dat** dành để nhập file **OutFile.dat** dành để xuất. Giả sử rằng cả hai file được đưa vào trong thư mục con có tên là **C:vb\project**.*

```
Open "C:\vb\project\InFile.dat" for Input as #1
Open "C:\vb\project\OutFile.dat" for Output as #2
```

7.5.1.1 Hàm FreeFile

*Như đã đề cập trước đây, mở một file, một người dùng có thể gán cho file bất kỳ số nguyên nào nằm giữa 1 và 511. Đây là một tác vụ dễ dàng lúc có một vài file phải xử lý. Tuy nhiên, lúc có một số lượng file lớn thì tốt nhất bạn nên sử dụng hàm FreeFile gán số có sẵn kế tiếp cho một file. Theo cách này người dùng tránh được những điểm bất tiện trong việc gán số giống nhau cho nhiều file cùng một lúc. Công dụng của hàm **FreeFile** được minh họa trong ví dụ kế tiếp.*

***VÍ DỤ 7.5** Hãy viết lại các câu lệnh của các ví dụ trên đây bằng cách sử dụng hàm FreeFile.*

```
Dim iInFile As Integer 'This variable is used to open the input file
Dim iOutFile As Integer 'This variable is used to open the output file
iInFile=FreeFile
Open "C:\vb\project\InFile.dat" for Input as #iInFile
iOutFile=FreeFile
Open "C:\vb\project\OutFile.dat" for Output as #iOutFile
```

Hàm FreeFile được gọi trước khi mở câu lệnh để tìm số có sẵn kế tiếp. Trong mỗi trường hợp giá trị được trả về bởi hàm FreeFile được gán cho một biến nguyên rồi nó được dùng trong câu lệnh mở.

7.5.1.2 Câu lệnh đóng

Lúc một chương trình đã xử lý xong một file thì file này cần phải

được đóng. Có hai lệnh có thể được dùng để thể hiện điều này

Close #filenumber *đóng file mà nó đã gán số file nguyên. Tất cả các file khác nếu có vẫn được mở.*

Close *đóng tất cả các file đã mở, lưu ý rằng trong trường hợp này chúng ta không cần chỉ định bất cứ số file nào.*

7.5.1.3 Đọc dữ liệu từ các file tuần tự

Như được đề cập trước đây, dữ liệu trong bất kỳ file tuần tự nào được đọc theo cùng thứ tự mà chúng được viết. Như ví dụ 7.6 đã chỉ rõ lúc một file tuần tự được mở, người dùng bắt đầu đọc từ đầu của file. Câu lệnh cho phép chúng ta đọc các record từ một file tuần tự có dạng như sau:

Input #Filenumber, field1, field2, ... fieldn

Ở đây Input là một từ khóa chỉ ra rằng chúng ta đang đọc (nhập vào) dữ liệu từ một file được mở trong chế độ Input. Filenumber là một số nguyên được gán cho file đã mở trước đây mà từ đó chúng ta đang đọc.

Lúc chuẩn bị một file tuần tự để nhập vào các chuỗi ký tự được đặt trong các dấu ngoặc kép và tách nhau bằng dấu phẩy. Các file tuần tự thường được tạo ra với một chương trình biên soạn hoặc với một chương trình xử lý văn bản. Tuy nhiên, lúc được tạo ra với một chương trình xử lý văn bản thì các file tuần tự cần phải được lưu dưới dạng text.

VÍ DỤ 7.6 *Giả sử rằng có một file tuần tự (models.dat) có chứa mẫu xe hơi khác nhau mà Silverado Motors hiện có trong kho. Hãy viết một thủ tục Visual Basic để đọc file tuần tự này và hiển thị nội dung trong một listBox có tên là lstModelsOnStock. Sử dụng file test minh họa dưới đây.*

```
Private Sub ReadModels()
 Dim stModels As String 'This variable is used to read the models in
 Dim iLoopIndex As Integer 'This variable is used as index of the for loop
 Open "C:\temp\Vbjunk\models.dat" For Input As #1
  For iLoopIndex=0 To 7
    Input #1, stModels
  DisplayForm.lstModelsOnStock.AddItem stModels, iLoopIndex
Next iLoopIndex
```

Trong ví dụ này chúng ta giả sử rằng listBox xuất hiện trong một dạng tên là DisplayForm. Lưu ý rằng phương pháp AddItem được dùng để điền vào listBox. Biến iLoopIndex phục vụ hai mục đích,

trước tiên là dùng để điều khiển việc thực thi vòng lặp. Thứ hai bởi vì chỉ số vòng lặp bắt đầu tại 0 cho nên nó cũng phục vụ làm một chỉ số cho danh sách. Một chỉ số danh sách trong Visual Basic bắt đầu tại 0. bởi vì có 8 chế độ trong file nhập, cho nên biến iLoopIndex thay đổi từ 0 cho đến 7 dạng tổng quát của lệnh để điền vào trong listBox có dạng như sau:

```
Object.AddItem value [,index]
```

Các dấu móc ngoặc vuông chỉ ra rằng chúng ta có tùy chọn để sử dụng một Index.

Mã trong ví dụ 7.6 giả sử rằng có chính xác 8 chế độ khác nhau trong file nhập. Bài tập có lời giải 7.2 biểu thị một phương pháp tổng quát hơn để đọc dữ liệu từ một file mà không yêu cầu phải biết rõ số các record trong file.

7.5.1.4 Viết dữ liệu và một file tuần tự

*Để viết dữ liệu vào một file tuần tự hãy sử dụng câu lệnh **Write**. Dạng tổng quát của lệnh này là:*

Write #Filenumber, field1, field2, field3, ... fieldn

*Ở đây **#Filenumber** là một file được mở trong chế độ **Output** hoặc **Append** và **field1, field2, ... filedn** là các file khác nhau mà chúng ta muốn viết vào file đó. chúng ta tham chiếu một cách tổng thể tất cả các trường này dưới dạng "danh sách các trường". Lúc **Visual Basic** viết các trường này vào đĩa, thì nó tách rời chúng bằng các dấu phẩy, dữ liệu String được đặt bên trong các dấu ngoặc kép và dữ liệu số thì được viết "as is". Sau khi viết phần tử cuối cùng của danh sách, **Visual Basic** tự động chèn một carriage trả về và một dòng **line feed** vào file.*

*Bảng 7.2 biểu thị hai chế độ khác nhau có thể được dùng để viết các record và một file: **Output** và **Append**. Lúc chế độ **Output** được dùng dữ liệu được viết vào file bắt đầu tại phần cuối của file này. Chế độ này viết chồng lên bất cứ dữ liệu nào đã có sẵn trong file. Lúc chế độ **Append** được dùng, dữ liệu được đứng vào ở cuối của file theo sau là một mảng thông tin cuối cùng được viết trước đó vào file.*

7.5.1.5 Hàm End Of File

Trong ví dụ 7.6 để thực thi chương trình trực tiếp chúng ta cần phải biết chính xác số các phần tử cần được đưa vào trong danh sách. Một phương pháp phổ biến hơn để điền vào danh sách đó là đọc các mẫu xe khác nhau cho đến khi bạn chạy hết các phần tử trong khi nhập. Nói cách khác chúng ta đọc các mẫu xe khác nhau cho

đến khi chúng ta đọc đến cuối file. Hàm ***EOF*** *cho phép chúng ta dò tìm lúc đã đọc được cuối file hoặc của hàm* ***EOF*** *như sau:*

EOF (Filenumber)

ở đây số file chính là số nguyên liên kết với một file mở trước đây.

Hàm ***EOF()*** *trả về một giá trị true lúc tìm thấy dấu kiểm cuối file. Điều này xảy ra khi dữ liệu cuối cùng trong file đã được đọc hoặc ngay khi chúng ta thử đọc từ một file có nguồn gốc là trống. Bài tập có lời giải minh họa cách sử dụng hàm* ***EOF.***

7.5.2 Các File trong C++

C++ xem các file như là một chuỗi các file mặc dù quan điểm này đã được chia sẻ bởi các nhà xuất trình biên soạn. Nhưng ở thời điểm sách này ra đời thì không có một tiêu chuẩn xác định nào về đặc trưng của hệ thống I/O. Trong mục này chúng ta sẽ thảo luận hệ thống I/O được sử dụng phổ biến nhất. Hệ thống này có tên là dòng C++ I/O dựa đặt cơ sở trên ba lớp khác nhau: istream để nhập, ostream để xuất và iostream để nhập xuất.

Các lớp này cho phép người dùng xác định và xử lý file chủ điểm các lớp sẽ được thảo luận chi tiết hơn trong chương 8.

Bất cứ lúc nào chúng ta khởi động chương trình C++, có biến lớp khác nhau được tạo một cách tự động. Những lớp này được chỉ định trong bảng 7.3.

Bảng 7-3

Biến	*Công dụng*
cin Console Input	*thường được gán cho bàn phím*
cout Console Output	*thường được gán cho màn hình*
cerr Console Erro	*thường được gán cho màn hình*
clog Console Log	*thường được gán cho màn hình*

Tất cả các biến lớp này được xác định trong file tiêu đề <isotream.h> điều quan trọng cần phải nhớ rằng chúng ta cần chọn phiên bản phù hợp hoặc chế độ phù hợp của lớp dòng lúc thực hiện đĩa I/O. Các lớp dòng I/O được chỉ định trong bảng 7.4 được đưa vào trong file tiêu đề I/O stream.

Bảng 7-4

Lớp	Công dụng
Istream	Input File Stream/ biểu thị một dòng các ký tự đến từ một file nhập
Ofstream	Output File Stream/ biểu thị một dòng các ký tự xuất ra khỏi một file xuất
Fstream	File Stream/biểu thị một dòng I/O đi ra và đi vào một file (file ngẫu nhiên)

7.5.2.1 Các khai báo các dòng file

Trong C++ được thiết lập để khai báo các biến của dòng trước khi sử dụng chúng. Ví dụ sau đây minh họa điều này. Luôn luôn phải đưa vào tiêu đề fstream.h bất cứ lúc nào bạn thực hiện đĩa I/O.

***VÍ DỤ 7.7** Hãy viết các phần khai báo C++ dành cho các file InputFile và OutputFile giống như tên của chúng đã chỉ ra, các file sẽ được dùng để nhập và xuất dữ liệu tương ứng.*

```
#include <fstream.h>  //this header needs to be included to do disk I/O
:
ifstream InFile;      //declaration of the input stream variable
ofstream OutFile;     // declaration of the output stream variable
```


Phần mã này khao báo hai biến dòng một trong số chúng là InFile sẽ được liên kết với một file nhập có nghĩa rằng một file mà từ đó chúng ta sẽ đọc dữ liệu. Cũng vậy biến OutFile cũng liên kết với một file xuất có nghĩa là một file mà chúng ta sẽ viết hoặc đinh một vài dữ liệu.

7.5.2.2 Mở và đóng các file trong C++

Việc mở một file trong C++ phục vụ cho hai mục đích. Trước tiên nó liên kết với một file thể lý với một biến dòng được xác định trong chương trình. Thứ hai phụ thuộc vào chế độ của file (file nhập và file xuất) dấu kiểm đọc được xác lập ở phần đầu hoặc đến cuối của file.

***VÍ DỤ 7.8** Hãy viết các lệnh cần thiết để mở các file được xác định trong ví dụ trước đây. Bằng cách sử dụng khai báo trước đây và được lập lại để thuận lợi cho người đọc, chúng ta có hai biến dòng.*

***ifstream** InFile;*

***ofstream** OutFile;*

Để mở những file này chúng ta có thể sử dụng hàm mở như minh họa dưới đây:

*InFile.**open** ("externalInpoutFile.dat")*

*OutFile.**open** ("externalOutpoutFile.dat")*

Tên file mà chúng ta chuyển qua dưới dạng một tham số cho hàm mở cần phải được đặt trong dấu móc kép. Lưu ý rằng thậm chí mà chúng ta sử dụng hàm mở giống nhau nhưng lệnh đầu tiên phải được liên kết với ifstream trong khi lệnh thứ hai phải được liên kết với ofstream. Quan sát để sử dụng chú thích là dấu chấm lúc gọi hàm mở.

Cả hai lệnh sẽ xác lập các dấu kiểm file tương ứng. Đối với file nhập, thì dấu kiểm đọc được lập ở phần đầu của file để cho thấy rằng file này đang hiện diện. Nếu file nhập không hiện diện, một lỗi sẽ xảy ra. Nếu file xuất cũng hiện diện thì dấu kiểm đọc được xác lập ở cuối file. Còn nếu file này không có thì hàm mở tạo ra một file mới và dấu kiểm đọc sẽ được xác lập ở đầu file này. Chuỗi được đặt trong các dấu móc (quotation) trong các câu lệnh mở để chỉ ra vị trí và tên của file trên đĩa. Nếu chuỗi đó không đưa vào bất cứ nội dung nào ngoại trừ tên file, thì file được tạo ra trên thư mục hiện tại.

Sau khi xử lý file chúng ta cần phải kết thúc chúng. Để sử dụng điều này chúng ta sử dụng các câu lệnh

InFile.close();

OutFile.close();

Để thực thi hai câu lệnh này sẽ giải phóng các nguồn được dùng bởi chương trình khác.

7.5.2.3 Sử dụng các file trong các câu lệnh Input và Output

Lúc làm việc với các file chúng ta vẫn có thể sử dụng các cấu trúc giống như đã từng dùng lúc làm việc với cin và cout như minh họa trong ví dụ kế tiếp.

***VÍ DỤ 7.9** Sử dụng file của hai ví dụ trước đây để nhập số lần mà một vòng lập cần tiếp phải được thực thi và xuất các giá trị khác nhau của trị số vòng lập.*

```
#include<iostream.h>
#include<fstream>
void main()
{ int i;
  int loopIndex;       /
  ifstream inFile ;    //declaration of the input file
  ofstream outFile;    //declaration of the output file
  inFile.open("c:/temp/data.dat");   //location of the input file
  outFile.open("c:/temp/data.out");  //location of the output file
  inFile>>i; //read the upper limit of the loop index
  for (loopIndex=1; loopIndex<=i; loopIndex++)
            outFile <<loopIndex<<endl;
  inFile.close();
  outFile.close();}
```

Lưu ý rằng trong đặc trưng của đường dẫn dành cho cả hai file nhập và xuất chúng ta đã sử dụng ký tự "/" thay vì sử dụng ký tự "\". Nếu chúng ta không sử dụng điều này thì bộ biên soạn sẽ không tạo ra một cảnh báo, bởi vì trong họ các ngôn ngữ giống như C thì ký tự "\" được dùng như là ký tự thoát.

7.5.3 Các file trong C

Để tham chiếu một file trong C người dùng cần phải khai báo một biến của kiểu file. Dạng tổng quát dành cho các biến khai báo của kiểu này là

FILE * variable_name;

ở đây FILE là một kiểu cấu trúc dữ liệu được chứa trong một file tiêu đề chuẩn stdio.h có chứa thông tin về trạng thái hiện tại của file. Variable_name ám chỉ đường tên của con trỏ mà người dùng chọn.

***VÍ DỤ 7.10** Khai báo hai biến file có tên là InFile và OutFile.*

Theo giả định dạng trên đây thì hai biến này có thể được khai báo như sau:

FILE * InFile;
FILE * OutFile;

*Ví dụ này giả sử rằng **stdio.h** được đưa vào phần đầu của chương trình.*

7.5.3.1 Mở các file trong C

Trong C, các lệnh file mở phục vụ hai mục đích, trước tiên nó xác lập một liên kết thể hình giữa chương trình và file dữ liệu, thứ hai nó đánh giá bằng nhau tên bên ngoài của file với biến con trỏ được khai báo trong chương trình.

Để mở một file trong C chúng ta sử dụng hàm fopen() hàm này như đã được chỉ định dưới đây, hàm này nhận hai đối số và trả về một con trỏ cho file nếu hoạt động thành công, nếu file không thể được mở, thì fopen trả về một NULL. Một tên file có thể đưa vào hoặc không đưa vào đường dẫn toàn bộ đến file đó.

fopen("filename", "acess moder");

Chế độ truy cập chỉ ra kiểu hoạt động mà người dùng dự định thực hiện trên một file. Bảng 7.5 minh họa tập hợp các tùy chọn hoàn chỉnh để truy cập một file. Trong sách này chúng ta chỉ xem xét các tùy chọn r, w và a.

VÍ DỤ 7.11 *Sử dụng các khai báo của ví dụ trên đây, hãy mở file InFile để đọc và mở file OutFile để viết. Giả sử rằng các vị trí và tên của các file nhập và xuất là "C:/temp/inData.dat" và "C:/temp/OutData.dat" tương ứng.*

Chúng ta có thể mở các file này như sau:

```
InFile=fopen("C:/temp/inData.dat","r");
OutFile=fopen("C:/temp/OutData.dat","w");
```

Bảng 7-5

Chế độ	*Công dụng*
r	*Mở file chỉ dành cho hoạt động đọc*
w	*Mở file để viết. Nếu file này đã có rồi thì nội dung của nó bị xóa. Ngược lại thì file này tạo*
a	*Mở một file để đính kết. Nếu file này không hiện diện thì nó được tạo ra.*
r+	*Nếu file này đã có sẵn thì nó mở cho mục đích đọc và viết nếu không nó trả về một lỗi.*
w+	*Tạo ra một file và mở nó dành cho mục đích đọc và viết. Nếu file này đã hiện diện thì nội dung của nó bị xóa.*
a+	*Mở file để đọc và viết nếu file này đã hiện diện. Nếu không thì nó tạo ra một file mới.*

7.5.3.2 *Sử dụng các file trong một lệnh nhập và xuất*

Một khi các file đều được mở chúng ta có thể sử dụng các hàm fscan(), fgets(), hoặc fgetc() để đọc file. Ngoài ra chúng ta vẫn có thể sử dụng các hàm fprintf(), fputs(), hoặc fputc() để viết vào các file.

Trong phần còn lại của chương này chúng ta sẽ sử dụng hàm fscan và fprintf để thực hiện các hoạt động I/O bởi vì chúng linh hoạt hơn các hàm fgets, fgetc, fputs hoặc fputc. Dạng tổng quát của fscanf và fprintf như sau:

fscanf(pointer_toi_file,format_string,&arguments)

fprintf(pointer_toi_file,format_string,arguments)

VÍ DỤ 7.12 *Bạn có một file nhập có chứa dữ liệu như minh họa dưới đây, hãy viết các câu lệnh khi cần thiết để đọc 4 record từ file nhập và viết các record này vào một file xuất khác tương ứng với dạng đã cho.*

Input file (indata.dat)	Output File (outdata.dat)
Smith Joe	Joe Smith
Randall George	George Randall
Weaver Earl	Earl Weaver
Ford James	James Ford

Trong chương trình này chúng ta giả sử rằng cả họ và tên đều giới hạn tối đa là 20 ký tự trong mỗi thứ chúng ta sử dụng hàm fscanf() để đọc các chuỗi từ file nhập và hàm fprintf() để viết các chuỗi sang file xuất. Cũng cần lưu ý rằng để chỉ định đường dẫn cho hai file nhập và xuất chúng ta phải sử dụng các dấu "/" đằng trước thay vì dấu "\" bên cạnh đó chúng ta cũng phải thoát chương trình nếu có một thông báo lỗi lúc mở file.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{ int loopIndex;
 char Fname[20], Lname[20]; //arrays to hold first and last names
 FILE *inFile;
 FILE *outFile;
 outFile=fopen("c:/temp/outdata.out","w");//open file for output
 /*open file for input. If there is an error, exit program*/
 if ((inFile=fopen("c:/temp/indata.dat","r"))==NULL)
 { printf("\nFailed to open input file");
  exit (1); }
/*Read strings from the input file and write them to the output file*/
 for(loopIndex=1; loopIndex<=4; loopIndex++)
 { fscanf(inFile,"%s %s",Fname,Lname);
  fprintf(outFile,"%s %s\n",Lname,Fname); }
} //end of program
```

SOLVED PROBLEMS
Các bài tập có lời giải

7.1 Write a Visual Basic procedure that saves in a file whatever the user types in a text box. The text is saved to the file when user presses the Enter key. Assume that all inputs are saved to a file located in the temp folder of the C drive. All messages are written using characters of the English alphabet. Assume also that the file is opened at the form load event. Show the code to open the output file.

We can use the KeyPress event of the text box to detect when the user has pressed the Enter key. The integer input parameter KeyAscii of the KeyPress event contains the ASCII value of the character representing the pressed key. The KeyPress event only takes effect (triggers) if the key pressed by the user represents an ASCII character. To know exactly when the user has pressed the Enter key we compare the value of the input parameter KeyAscii with the Visual Basic predefined constant vbKeyReturn. This constant represents the ASCII code for the Enter key. Assuming that the text box is called txtInputBox, the code for this procedure may look as shown below. The skeleton for this procedure is automatically created by Visual Basic simply by double clicking on the textbox.


```
Private Sub TxtInputBox_KeyPress(KeyAscii As Integer)
   If KeyAscii=vbKeyReturn Then
      Write #1, txtInputBox.Text
   End If
End Sub
```

It is fairly common for files to be opened at the form load event. The code to open the file is shown below. Notice that we have used the FileFree function to obtain the next available file number. Since the output file is opened in append mode the file retains its previous content.

```
Private Sub Form_Load()
   Dim iFileNumber As Integer
      iFileNumber=FreeFile
      Open "C:\temp\OutFile.txt" For Append As #1
End Sub
```

7.2 Generalize the code of the Visual Basic procedure shown in Example 7.6 to allow the procedure to read from the input file until all its content has been exhausted.

```
Private Sub Readmodels()

    Dim stModels As String 'This variable is used to read the
    models in

    Dim iFileNumber As Integer

    iFileNumber=FreeFile

    Open "C:\temp\Vb\Projects\models.dat" For Input As
    #iFileNumber

        Do Until EOF(iFileNumber) Input #1, stModels

              DisplayForm.lstModelsOnStock.AddItem
              stModels, iLoopIndex

        Loop

End Sub
```

To allow the procedure to read from the models.dat file until all its values have been exhausted, we will use the EOF function. This function returns a value of True when the end of file has been detected. This occurs right after the last record has been read. Since the condition of the Do Until is evaluated at the top of the loop, the instruction comprising the body of the loop is not executed after the last record is read. The code of Example 7.6 was also modified to illustrate the use of the variable iFileNumber when testing for the EOF condition.

7.3 Write a Visual Basic Procedure that reads string values from an input file called C:\notes.txt. Include in the procedure the necessary instructions to detect errors like "drive is not available" or "file does not exist."

In Visual Basic, to detect if an error has occurred when opening a file we can use the **On Error instruction.** Visual Basic provides an Err object that contains information about the current error. The number that identifies the error is saved in the Err.number property. To determine the type of run-time error that has occurred and to take the appropriate actions, we need to intercept or trap the error. To do this we need to:

(1) Enable the error-handling feature using the On Error instruction. This instruction specifies the line that will be executed if an error occurs. That is, the line that the program will "go to" if there is an error. The format of this instruction is

On Error Goto labeled line

(2) Create a labeled line that marks the beginning of the error handler. A labeled line is a name followed by a colon. Nothing else should appear in this line.

(3) Write the error handler. That is, the code that will be executed

if an error occurs.

(4) Determine if the program is to continue executing after the error. If so, determine how and where execution should start.

```
Private Sub Form_Load()
   Dim iFileNumber As Integer
   Dim stLine As String
   Dim ErrorMessage As String
iFileNumber=FreeFile
On Error GoTo HandleError 'This instruction corresponds to
   step 1 above
Open "C:\temp\Notes.txt" For Input As #1
Do Until (iFileNumber)
Input #iFileNumber, stLine
   Loop
HandleError: 'labeled line that corresponds to step 2 above
   Select Case Err.Number 'Here begins the error handler of
   step 3
   Case 53
      ErrorMessage="File does not exist"
   Case 68
      ErrorMessage="Drive is not available"
   Case 71
   ErrorMessage="Disk is not ready"
   End Select
MsgBox ErrorMessage, vbInformation, "Missing File"
Exit 'Instruction that indicates next action to follow after
   error has been detected.
   'It corresponds to step 4
End Sub
```


The Select statement shows the value of some of the predefined Visual Basic constants that we can use to introduce minimal error-handling capabilities into the program. The user should consult the appropriate manuals if other cases need to be considered.

7.4 Write the necessary Visual Basic instructions to read the fifth record from the random file C:\Data\employee.dat and increase the salary of the **fifth** employee by $100.00. Assume that each

employee record has the following structure.

Last Name 15 characters

First Name 15 characters

Department 5 characters

Salary Number

Since all the records in a random file have the same length, it is necessary to set up the structure of the record using the **Type** and **Type** End statements. After the record structure has been set up, we can declare a record variable of that type.

The structure of the employee record indicated above can be defined as follows:

```
Private Type EmployeeRecord
    stLastName As String*15
    stFirstName As String*15
    stDepartment As String*5
    cSalary As Currency
End Type
Dim mEmployee As EmployeeRecord
```


The variable mEmployee has been declared to be of an EmployeeRecord type. The prefix letter m indicates that the variable is a member of the EmployeeRecord structure.

To specify that the string fields have a fixed length we have followed each string declaration with an asterisk and an integer number. This integer number indicates the maximum number of characters that the string can hold. Whenever the content of a field does not fill the entire string, Visual Basic adds extra blanks (pads the field) to make the content of the field fit the length of the string. If the number of characters assigned to a string variable is longer than the maximum number of characters that the variable can hold, Visual Basic will truncate (chop off) the extra characters.

To open the file we can use the same instruction that we used for opening sequential files except that now we also have to include the length of the records. The latter action is required for all random files. The instruction to open the file is

Open "C:WataXemployee.dat" For Random As #1 Len = Len(mEmployee)

where the right-hand side of the expression Len = Len(mEmployee) is a call to the Length function. This function returns the length

(in bytes) of the record mEmployee.

To read the records from the random file we can use the Get instruction. The format of this instruction is

Get #Filenumber, [RecordNumber], RecordName

where the square brackets indicate that we have the option of specifying the number of the record that needs to be read. Records numbers start at record 1.

To write a record to the random file we use the Put instruction. The format of this instruction is

Put #Filenumber, [RecordNumber], RecordName

where the square brackets indicate that we have the option of specifying the number of the record that needs to be written. Records start at record 1.

We now have all the elements to read the record of the fifth employee and increase the salary by $100.00. The instructions to do this may look like this:

```
Get #iFileNumber, 5, mEmployee //read the fifth record
mEmployee.cSalary=mEmployee.cSalary+100.00 //increase sal-
   ary
Put #iFileNumber,5,mEmployee //write fifth record back to
   file
```


Notice that to access the cSalary field we have to qualify the field by preceding it with the record name and a period. In this set of instructions, we have made the assumption that the file is not empty and that it has been opened successfully.

7.5 How can we determine the end of a random file in Visual Basic?

In a random file, we determine the end of the file indirectly through the use of the function LOF (Length Of File). This function returns the length of the file in bytes. If we divide the value returned by LOF by the length of a single record, we can calculate the highest record number in the file. We can use this value and compare it with the number of the current record being processed to determine if we have already reached the last record of the file. The formula to calculate the highest record in a random file is

Number of records in file = LOF(Filenumber)/Len(variable)

7.6 Assume that a sequential file contains information about the different employees of a company. The order in which the information appears in the file (C:\Dta\emp.dat) and some sample data are shown below. Write a Visual Basic procedure that reads from this sequential file and separates the employees according to the department in which they work. Each department should have its

own output file.

SSN	Name	Department	Salary
512-23-0987	Alice McPearson	Accounting	45000
345-90-1234	Andrew Antonelli	Marketing	32000
123-56-7898	Darlene Bartlow	Sales	30000
890-78-2345	Benjamin Nichols	Accounting	38000
340-34-1234	Mark Felton	Sales	35000

The input file can be prepared with any text editor. Each record in the input file (C:\Data\emp.dat) should have the following format:

"512-23-0987", "Alice McPearson", "Accounting", 45000.

A procedure to separate the employees into the different departments is shown below. This procedure uses four private variables, one for each of the fields that we want to read from the input sequential file. In addition, the procedure, in each of the write statements, makes use of a global variable that contains the integer number associated with a department output file. As the employees are read in, they are separated into their different departments through the use of a Select statement. The procedure to open the input and output files is also shown below.


```
Private Sub SeparateEmployeesIntoDepartments()
   Dim stSSN As String
   Dim stName As String
   Dim stDepartment As String
   Dim cSalary As Currency
   Do Until EOF(1)
      Input #1, stSSN, stName, stDepartment, cSalary
      Select Case stDepartment
            Case "Accounting"
                  Write #iAccountingFile, stSSN, stName,
   cSalary
            Case "Marketing"
                  Write #iMarketingFile, stSSN, stName,
   cSalary
            Case "Sales"
                  Write #iSalesFile, stSSN, stName, cSalary
```

```
        End Select
    Loop
End Sub
    Private Sub OpenDataFiles()
        Open "C:\temp\emp.dat" For Input As #1
        imarketingFile=FreeFile 'Get file number for the
        marketing file
        Open "C:\temp\mrktng.dat" For Output As #iMarketingFile
        iSalesFile=FreeFile 'Get file number for the sales file
    Open "C:\temp\sales.dat" For Output As #iSalesFile
    iAccountingFile=FreeFile 'Get number for the accounting de-
    partment
    Open "C:\temp\acctng.dat" For Output As #iAccountingFile
End Sub
```

7.7 What does the Visual Basic procedure shown below do? Assume the existence of the following user-defined data type.

```
Type EmployeeRecord
    LastName as String * 20
    FirstName as String * 20
    Salary As Currency
End Type
    Private Sub GuessWhatIdo()
    Dim iFileNumber As Integer
    Dim Employee As EmployeeRecord
    iFileNumber=FreeFile
    Open "C:\temp\emp.dat" For Random As #iFileNumber
    Len=Len(Employee)
    Employee.LastName="Scott"
    Employee.FirstName="George"
    Employee.Salary=23000 Put #iFileNumber, 10,
    Employee
End Sub
```

The code of this procedure writes to the random file C:\temp\emp.dat an employee record in the loth position of the file. This new employee record is for George Scott who has a salary of 23,000.

7.8 Assume that there is a sequential file that contains character strings

(one per line) that are preceded or followed by an unknown number of blanks. Write a Visual Basic program that eliminates these extra blanks surrounding each string and copies the resulting string to a new file.

The execution of this program is controlled by the procedure Main. Since there is no form associated with this program a Main procedure is mandatory. Automatically Visual Basic starts the execution of the program with this procedure.

Each of the procedures that comprise the program performs a single function.

The procedure PrOCeSSFdes reads the input file, calls the procedure to eliminate the blanks and writes to the output file. In this procedure, we have used the function EOF to read the input file until all its content has been exhausted.

The procedure EliminateSurroundingBlanks makes use of the TRIM(string) function to get rid of the leading and trailing blanks of any string. The TRIM() function preserves any blanks embedded in the strings. Observe the use of the input parameter in the assignment statement where the TRIM() function is used. Since the parameter is passed by reference, any changes to this parameter are immediately reflected in the actual parameter.

The procedure CloseFiles does exactly what its name indicates: it closes the files. In this procedure, we have closed the files individually. However, we could have closed both files by means of the Close instruction.

```
Option Explicit
   Dim inFile As Integer
   Dim outFile As Integer
Private Sub EliminateSurroundingBlanks (stInString As
   String)
   stInString=Trim(stInString)
End Sub
Private Sub OpenFiles()
   inFile=FreeFile
      Open "C:\temp\blanks.dat" For Input As #inFil'e
   outFile=FreeFile
      Open "C:\temp\noblanks.dat" For Output As #outFile
End Sub
Public Sub Main()
   OpenFiles
```

```
    ProcessFiles
    CloseFiles
End Sub
Private Sub ProcessFiles()
    Dim stLine As String
    Do Until EOF(inFile)
        Input #inFile, stLine
        EliminateSurroundingBlanks (stLine)
        Write #outFile, stLine
    Loop
End Sub
Private Sub CloseFiles()
    Close #inFile
    Close #outFile
End Sub
```

7.9 Assume that a file contains a series of character strings. Each string can be up to 80 characters long. Each string is a sequence of digits and letters from the English alphabet. Each string occupies one line. Write a C++ program that reads the strings from the file "C:/temp/sequence.dat" and counts the number of letters and digits present in each string. Write the output of this program to the file "C:/temp/count.dat".

This program consists of four functions: main, openfiles, countcharacters and closefiles. The role of the main function is to call the other three functions in sequence. The function openfiles does exactly that. It opens the files for reading and writing. Likewise, the function closefiles closes all files that were opened in the program. The countcharacters function reads one line at a time from the input, file and examines each character to see if it is a letter or a digit. The funcion isalpha() is used to determine if the current character is a letter or not. Likewise, the function isdigit() is used to determine if the current character is a digit. If the current character is a letter or a digit, the corresponding counters are incremented. After examining the characters of the input line, the program writes each line to the output file.

```
#include<iostream.h>
#include<fstream.h>
#include <stdlib.h>
#include <ctype.h>
```

```
void   openfiles(void);  //function prototypes
void   countcharacters(void);
void   closefiles(void);
const int Line_Size=80;     //variables that define the input
   line
char   InLine[Line_Size];
ifstream InFile;    //file variable declaration
ofstream OutFile;
void main()
{ openfiles();
   countcharacters();
   closefiles();
}
void openfiles(void)
{
InFile.open("c:\temp\sequence.dat");
if (InFile.bad())
{  cerr<<"Error: Could not open input file\n"; exit(8); y
OutFile.open("c:\\temp\\count.dat"); if (OutFile.bad())
   {   cerr<< "Error: Could not open output file\n";
       exit(1); }
}
void countcharacters(void)
{  int i, digits, letters;
   while(!InFile.eof())
   {   digits=0;
       letters=0;
       InFile.getline(InLine,sizeof(InLine),'\n');//
       read input line
       for(i=0; i<Line_Size; i++)
       {      if (isalpha(InLine[i])) // Is character a let-
   ter?
              letters++;
              else
                     if (isdigit(InLine[i])) //Is character a
   digit?
                     digits++;
```

```
        }
    i=0;
OutFile << "Input Line:"; //write line to output file
while(InLine[i]!= " )
{  OutFile<<InLine[i];
   i++;
   }
   OutFile<<endl <<"Letters="<< letters;
   OutFile<<endl<<"Digits="<<digits<<endl;
   OutFile<<endl;
}
   OutFile<< endl;
}
void closefiles(void)
{
InFile.close();
OutFile.close();
}
```


7.10 Write a C++ procedure that reads a student name and his class grades from the file C:\exams.dat. Write the student's name and his average to an output file C:\grades.dat. Use the input sample data and output guidelines indicated below.

Sample Input Data	Sample Output Data
Anita Jones 89 92 75 92	Anita Jones 87
Joseph Martin 82 89 93 91	Joseph Martin 88.75
Albert Smith 83 82 90 91	Albert Smith 86.5

The C++ program may look like the one shown below. There are three functions in addition to the main function. The openfiles and closefiles functions perform the tasks indicated by their names. The calculategrades() function reads the grades of each student and determines the corresponding average.

```
#include<fstream.h>
#include <stdlib.h>
void openfiles (void); void calculate_grades(void);
const int totalexams=4; float exams [totalexams];
char firstname[10]; char lastname[10];
```

```
ifstream InFile; ofstream OutFile;
void main()
{
   openfiles();
   calculate_grades();
   closefiles(0;
}//end of main void openfiles(void)
InFile.open ("c:\\temp\\exams.dat");
if (InFile.bad())
   {
   cerr« "Error: Could not open input file\n";
   exit (8);
   }
OutFile.open("c:\\temp\\grades.dat");
if (OutFile.bad())
   {
      cerr« "Error: Could not open output file\n";
      exit (1);
   }
}//end of openfiles
void calculate-grades (void)
{  int i ;
   float average, sumofgrades;
   while(!InFile.eof())
   {
      InFile»firstname»lastname; //read student's first and
   last names
      sum=0;
      for(i=0; i <totalexams; i++)
      {
      InFile»exams[i]; //read grades
      sumofgrades=sumofgrades+exams[i]; //keep running to-
   tal
      }
   average= sumofgrades/totalexams; //calculate average
   OutFile«firstname«1' 11<<lastname « " "<<average «endl;
   }
}//end of calculate_grades
void closefiles(void)
{
   InFile.close();
   OutFile.close();
}
```

7.11 Write a C++ program that reads a string of data from an input file and replaces every vowel in the string with some other characters according to the substitution rules indicated below.

Substitute:

a,A for ? e,E for * i,I for # o,0 for % u, U for ^

The program to accomplish this task may look like the one shown below. The program, in addition to the main function, has three functions that perform specific tasks. The openfiles function does exactly what its name indicates. It opens the input and output files and notifies the user of any error that may occur while opening these files. Likewise, the closefiles function closes the input and output files. The Convertdata function reads a line and replaces each vowel in the line according to the given specifications. Notice the use of the switch-case construct to change each of the vowels. The clearline function is necessary to erase any leftover characters of any previously read line that happens to be longer than the current line.

```
#include<fstream.h>
#include<stdlib.h>
void openfiles (void); //function prototypes
void convertdata(void); void clearline(void);
void closefiles(void);
const int maxlength=80; //This variable will hold the input
   lines read from the file char line [maxlength];
ifstream InFile;    //file variable declaration
ofstream OutFile;
void main()
{
   openfiles();
   convertdata();
   closefiles();
}  //end of main
void openfiles(void)
{
   InFile.open ("c:\\temp\\convdata.dat");
   if (InFile.bad())
   {
       cerr« "Error: Could not open input file\n";
       exit (8);
```

```
    }
    OutFile.open("c:\\temp\\convout.dat");
    if (OutFile.bad())
    {
        cerr<<"Error: Could not open output file\n";
        exit (1);
    }
} //end of openfiles
void convertdata(void)
{  int i;
   while(!InFile.eof())
{
InFile.getline(line,sizeof(line),'\n');
OutFile<<"In: "<<line<<endl;
OutFile<< "Out:";
for(i=0; i< maxlength; i++) //Is character a vowel?, if so,
   change it
{
   switch (line[i])
       {
   case 'a':
   case 'A':
   line[i]='?';
   break;
   case 'e':
   case 'E':
   line[i]='*';
   break; case 'i':
   case 'I':
   line[i]='#';
   break;
   case 'o':
   case '0':
   line[i]='$';
   break;
```

```
        case 'u':
        case 'U':
        line[i]='^';
        break;
            }// end switch
        OutFile<<line[i];
    }//end for
    OutFile<<endl;
    clearline();
        }//end of while
    }//end of convertdata
    void clearline(void)
    {   int i;
        for (i=0; i<maxlength; i++)//replace any existing char-
        acter with a blank
        line[i]=' ';
    }//end of clearline
    void closefiles(void)
    {
        InFile.close();
        outFile.close();
    }//end of closefiles
```

7.12 **Write a C++ program that reads a line from an input file and writes the line backward to an output file. Make sure that the input and its corresponding "backward" line are shown in the output file.**

```
#include<fstream.h>
#include<stdlib.h>
void openfiles(void);
void printbackward(void);
void clearline(void);
void closefiles(void);
const int maxlength=80;
char line[maxlength];
ifstream InFile;
```

```
ofstream OutFile;
void main()
{  openfiles();
   clearline();
   printbackward();
} //end of main
void openfiles(void)
{
   InFile.open ("c:\\temp\\inline.dat");
   if (InFile.bad())
   {
      cerr<<"Error: Could not open input file\n";
      exit (8);
   }
   OutFile.open("c:\\temp\\outline.dat");
   if (OutFile.bad())
   {
      cerr<< "Error: Could not open output file\n";
      exit (1);
   }
}//end of openfiles
//This procedure starts copying from the last character in
   the input line and moves toward the front of the array.
void printbackward(void)
{ int i;
   while(!InFile.eof())
   {
      InFile.getline(line,sizeof(line),'\n');
      OutFile<<"In: "<<line<<endl;
      OutFile<< "Out:";
      for(i=maxlength-1; i >=0 ; i--)
      { OutFile<<line[i];
      }//end for
      OutFile<< endl;
      clearline();
```

```
    }// end of while
}// end of printbackward
void clearline(void) // replace existing characters with blanks
{ int i ;
for (i=0 ; i< maxlength; i++)
    line[i] = ,
}
void closefiles()
{
    InFile.close();
    OutFile.close();
} // end of closefiles
```

7.13 Write a C++ program that reads a string line from a file and determines if the input line is a palindrome. Assume that every string is a sequence of characters with no intervening blanks. Use the following sample data and make sure that the input characters are all of the same case. That is, all uppercase or all lowercase.

Sample Data

RADAR
AREPERA
11111111
11113333

The procedure palindrome locates the first nonblank character starting from the last character of the input line. The location of this nonblank character is then used as the starting-point for the comparison of the string characters. Notice that it is necessary

```
#include<fstream.h>
#include<stdlib.h>
#include<ctype.h>

void openfiles (void);
void palindrome(void);
void clearline(void);
void closefiles(void);

const int maxlength=80;
char line [maxlength];

ifstream InFile;
ofstream OutFile;
```

```
void main()
{
  openfiles();
  clearline();
  palindrome();
}//end of main
void openfiles(void)
{
    InFile.open ("c:\\temp\\inline.dat");
  if (InFile.bad())
   {
         cerr<<"Error: Could not open input file\n";
         exit (8);
   }
  OutFile.open("c:\\temp\\outline.dat");
  if (OutFile.bad())
  {
         cerr<<"Error: Could not open output file\n";
         exit (1);
  }
}//end of openfiles
void palindrome(void)
{ int i,j,k;
  while(!InFile.eof())
  {
    InFile.getline(line,sizeof(line),'\n');
    OutFile<<"In: "<< line<<endl;
    /* find first nonblank character starting from the
      last character and moving to the first */
    for(i=maxlength-1; i<=0; i--)
         if (line[i] !=' ')
         {
         j=i;
         break;
```


to compare the characters in pairs. That is, we need to compare the first character of the string with the last; then the second character with the next to last and so on. This procedure is repeated until all the characters have been examined and all the pairs match or until there is a pair that does not match. The other procddures of this program are similar to the ones used in the previous solved problems.

```
    for( i=0; i<j; i++)
    {
      if (line[i] !=line[j-i])//set the flag k if the pair does not match
        { k=1;
          break;
        }
      )//end for
    if (k==1)
      OutFile<<'The input line is not palindrome'<<endl;
    else
      OutFile<<'The input line is a palindrome'<<endl;
      clearline() ;
    )//end of while
}//end of printbackward
void clearline(void)
{ int i;
  for (i=0 ; i<maxlength; i++)
      line[i]=' ';
}
void closefiles()
{
  InFile.close();
  OutFile.close();
}
```



```
k=0; //this variable is used to flag if the string is not a
   palindrome
j--;//adjusting variable to position of last character of
   //the string compare first and last character, then compare
   // second and next to last until all characters match
   //or there is a pair of characters that do not.
```

7.14 Is it necessary to verify that a file has been opened correctly before using it? Is there an easier way to verify that the file has been opened correctly?

It is always convenient to verify that a file has been opened successfully because, in general, the compilers of the C-like family may not report all errors in an accurate manner or may not report the error at all. If there is an error file, the program may not work at all and it may not be clear why the program failed. Another way to determine if a file has been opened successfully in C++ is through the use of the function assert(file variable). This function, which requires the inclusion of the header file <assert.h>, aborts the program in case of an error. Although this is not the most elegant way to terminate a program, it provides an easier mecha-

nism to verify that a file has been opened correctly. If a file opens successfully, the file variable, such as InFile or OutFile, evaluates to a nonzero value, otherwise the function returns a zero. This can be illustrated using the open statement of Solved Problem 7.13. The code to open the InFile needs to be modified as follows:

```
<assert.h> //This header needs to be included at the top of
   the program

ifstream InFile("c:\\temp\\inline.dat");//assert (InFile);
   // verify that the open file has not been opened correctly
:
:
of stream OutFile ("c:\\temp\\outline.dat"); assert
   (OutFile);
```

If there is an error opening the input file InFile, the compiler will generate an

```
void WriteToScreen(void)
{
 int const screensize=24;
 char inputline[80];
 int linecounter=0;
 ifstream InFile ("C:\\temp\\taxdata.dat");
 assert(InFile);
while (! InFile.eof())
{
 InFile.getline(inputline,sizeof(inputline),'\n');
 cout<<inputline<<endl;
 if ((++linecounter % screensize) == 0)
  {
    cout<<"--MORE--";
    cin.get();
  }
 }
InFile.close();
}//end of procedure WriteToScreen
```


error such as

Assertion failed: InFile, Me D:\backward\backwards.cpp, line 04

Some compilers give us the option of aborting the process, ignoring the error or retrying the statement to open the program.

7.15 Write a C++ procedure that reads an input file (C:\taxdata.dat) and displays its content "one screen at a time." Assume that there are 24 lines per screen.

7.16 Write a C program that reads a file containing a list of items and their prices. There is one item per line and each item is followed by its price.

To read from the input file, first, we need to declare a variable of type FILE as shown below. Once the file variable has been declared it is necessary to open the file. To do this we need to pass two parameters to the fopen function. The first parameter is the path and name of the file. The second parameter to the fopen function is the mode. Since we want to read a file we use the "r" mode. Following the fopen function the program tests that the file has been opened correctly. If the input file does not exist, the fopen function returns the NULL value. If an error occurs, the program displays an error message on the screen and the execution terminates. The program reads from the file until all its content has been exhausted. Notice how the program tests for the end of file (EOF) condition. To read each of the input lines the program uses the fscanf function. Notice that this function has three parameters: the filename, the descriptors (one per variable), and the variables that need to be read in. The %s descriptor reads strings separated by blanks, tabs, or new line characters. After the last line is read in, the program detects the EOF marker. The while

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  char item[15];
  float price;
  FILE *InFile;  //variable file
  InFile=fopen("C:/temp/items.dat","r");
  if(InFile==NULL)  //was the file opened successfully?
  {
     printf("\nCould not open input file\n");
     exit (1);
  }
  //read the file until its content is exhausted
  while(fscanf(InFile,"%s %f", item, &price)!=EOF)
    printf("%-17s %5.2f\n",item,price);
fclose(InFile);
}
```


condition fails after the EOF has been detected and no other line is read in from the input file. As the items and their prices are read in, the program displays them on the screen. To ensure that

the items and the prices are nicely separated, the output has been

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
char fileToOpen[15];
FILE *OutFile; //file variable
printf("Print the name of the file to open:"); //prompt user for file name
scanf("%s", fileToOpen); //read name typed by user
OutFile=fopen(fileToOpen,"w"); //file opened for writing
if(OutFile==NULL) //was the file opened successfully?
{
   printf("\nCould not open input file\n");
   exit (1);
}
fprintf(OutFile,"\nThe file was opened successfully");
fclose(OutFile); //close file
}
```

formatted so that the items are left-justified in an output field 17 characters long. Prices are displayed with two decimals. After the file has been read, the program finishes by closing all files.

7.17 Write a C program that prompts the user for a file name and then proceeds to open the file for writing. Exit the program if an error occurs while opening the file.

The code to implement this file may look like this:

7.18 Write a C program that accepts input lines typed by the user and saves them to an output file. The program finishes when the user types N to the question "Do you want to continue?"

The heart of the program is a continuous loop whose condition always evaluates to true. Within this loop, the user is prompted twice: the first time to enter an input line, the second time to check if he wants to continue inputting data. If the answer is no, the program finishes executing and notifies the user that it fin-

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  char Line[80]; //this variable holds the user input
   char answer; //this variable holds the answer typed by the user
   FILE *OutFile; // file variable
  OutFile=fopen("C:\\temp\\dataout","w"); // open file for output
```

```
    if( OutFile == NULL)
     {
       printf("\nCould not open output file\n");
       exit(1);
     }
   while ( 1==1 ) // Continuous loop. This condition always evaluates to true
     {
     printf("Enter input lines\n"); // prompt user
     gets(Line);
     fprintf(OutFile,"%s\n",Line);
     printf("Do you want to continue(y/n)? :"); //Continue executing the loop?
     answer=getchar(); // get single letter response
     if(toupper(answer) == 'N') // exit if user does not want to continue
     {
        printf("Program ended at user request\n");
        fclose(OutFile);
        exit(0);
     }
        answer=getchar(); // capture the new line (return) character
    }//end of while (1==1)
}//end of main
```

ished at his request. Notice that the responses of the user are

```
#include<stdio.h>
void main()
{
  int * OutFile;
  int loopindex;

  OutFile=fopen("c:\dta.dat");
  if( OutFile=NULL)
  {
     printf("\nCould not open output file\n");
     exit(1);
  }
  fputs (outFile, "What is wrong with me?");
}
```

converted to uppercase to account for the lower- or uppercase "n." This single-letter response is captured using the getchar() function. However, since this function only reads one character at a time, it is also necessary to capture the new line or "return" character generated when the user presses the Enter key after typing Y or N. This is the reason for calling the getchar() function again at the bottom of the while loop.

7.19 Is there anything wrong with this program?

Yes! There is. The program is missing #include<stdio.h>. If we do not include this the program will generate an error like

dAProject\lines.cpp(17) : error C2065: 'exit' : undeclared identifier. In addition to this, the file variable OutFile has been declared as int instead of type FILE and the open function is missing the mode. The condition of the if statement is incorrect. Since we are making a comparison, we should have used == instead of =. In the puts statement the name of the file has been misspelled. Remember that C is case-sensitive and that outFile is not the same as Oufle. A more subtle error is that of not closing the file.

7.20 Write a utility that copies one file to another and returns the number of characters copied.

```
long fileCopy (void)
{
   FILE *source;        /*source file pointer*/
   FILE *destination;   /*destination file pointer*/
   char nextchar;       /*next input character*/
   long charcount=-1L;  /*variable to count number of characters copied*/
   if ((source=fopen("c:\\project\\indata.dat","r"))==NULL)
    printf("Can't open source file for reading\n");
   else
   {
    if ((destination=fopen("c:\\project\\outdata.dat","w"))==NULL)
       printf("Can't open output file for writing\n");
    else
       {
         charcount=0L; //initialize counter
         while ((nextchar=getc(source)) != EOF)
         {
           charcount++;
           putc(nextchar,destination); //copy characters to destination file
         }
         fclose(destination);
       }
        fclose(source);
     }
  return charcount;
}//end of function
```

7.21 Write a C program that prompts the user for two files, compares these two files and determines if their contents are identical.

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
void CompareFiles(FILE*,FILE*);
const int MaxLength=80; // length of the input lines
void main()
{
  char firstfileToOpen[80];
  char secondfileToOpen[80];
```

```
FILE *FirstFile; //file variables
FILE *SecondFile;
printf("Enter first file to compare:"); //prompt user for file name
scanf("%s", firstfileToOpen); //read name typed by user
printf("Enter second file to compare:"); //prompt user for file name
scanf("%s", secondfileToOpen); //read name typed by user
FirstFile=fopen(firstfileToOpen,"r"); //file opened for writing
if(FirstFile==NULL) //was the file opened successfully?
{
   printf("\nCould not open first file\n");
   exit (1);
}
SecondFile=fopen(secondfileToOpen,"r"); //file opened for writing
if(SecondFile==NULL) //was the file opened successfully?
{
   printf("\nCould not open first file\n");
   exit (1);
}
CompareFiles(FirstFile,SecondFile);
   fclose(FirstFile); //close files
   fclose(SecondFile);
}
void CompareFiles(FILE* file1, FILE* file2)
{
   char line1 [MaxLength];
   char line2[MaxLength];
   char* ptrline1;
   char* ptrline2;
   ptrline1=fgets(line1,MaxLength,file1);
   ptrline2=fgets(line2,MaxLength,file2);
   if(feof(file1) !=0 | feof(file2) !=0)
   {
   printf("\nOne of the files is empty and cannot be compared\n");
   exit(0);
   }
else
{
   while (ptrline1 !=NULL && ptrline2 !=NULL)
   {
      if (strcmp(line1,line2) !=0)
      {
         printf("\nFound two different lines\n");
        exit(1);
      }
      if(feof(file1)==0)
        ptrline1=fgets(line1,MaxLength,file1);
      else break;
      if(feof(file2)==0)
        ptrline2=fgets(line2,MaxLength,file2);
```

```
            else break;
        }
        printf("\nFiles are identical\n");
        exit(0);
      }
    }//end of function CompareFiles
```

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

7.1 Hãy viết một thủ tục Visual Basic để lưu trong một file bất cứ lúc nào người dùng gõ nhập trong một text box. Text được lưu sang file lúc người dùng nhấn phím enter. Giả sử rằng tất cả các dữ liệu nhập đều được lưu một file nằm ở bên trong folder temp của đĩa C. Tất cả thông điệp được viết bằng cách sử dụng các ký tự theo mẫu tự bảng chữ cái tiếng Anh. Cũng giả sử rằng các file được mở theo dạng biến cố tải. Hãy biểu thị mã để mở file Output này.

7.2 Khái quát hóa mã của Visual Basic như minh họa trong hình 7.6 để cho phép thủ tục đọc từ file nhập cho đến khi tất cả các nội dung đều đọc hết.

7.3 Hãy viết một thủ tục Visual Basic cần thiết để đọc các giá trị chuỗi từ một file nhập có tên là C:\notes.txt. Đưa vào trong thủ tục các lệnh cần thiết để dò tìm các lỗi chẳng hạn như "drive is not available" hoặc "file does not exist".

7.4 Hãy viết các câu lệnh Visual Basic cần thiết để đọc record thứ năm từ file ngẫu nhiên C:\Data\employee.dat và tăng lương của công nhân thứ năm một lượng là 100 đôla. Giả sử rằng mỗi hồ sơ của nhân viên đều có cấu trúc sau đây.

7.5 Bằng cách nào chúng ta có thể xác định phần kết thúc của một file ngẫu nhiên trong Visual Basic.

7.6 Giả sử rằng file tuần tự có chứa thông tin về các nhân viên khác nhau trong một công ty. Thứ tự qua đó thông tin xuất hiện trong file (C:\Dta\emp.dat) và một vài file mẫu được minh họa dưới đây. Hãy viết một thủ tục Visual Basic để đọc từ file tuần tự này và tách rời các nhân viên theo phòng ban mà chính họ làm việc. Mỗi phòng ban có một file xuất riêng.

7.7 Thủ tục Visual Basic nào được minh họa dưới đây? Giả sử rằng có sự hiện diện của chuỗi dữ liệu mà người dùng định nghĩa.

7.8 Giả sử rằng có một file tuần tự có chứa các chuỗi ký tự (mỗi chuỗi trên một dòng) đứng trước hoặc đứng sau một số các hoạt động chưa được biết. Hãy viết một chương trình Visual Basic để loại bỏ những khoảng trống bao quanh các chuỗi này và sao chép chuỗi kết quả sang một file mới.

7.9 *Giả sử rằng một file có chứa một loạt các chuỗi ký tự, mỗi chuỗi có chiều dài lên đến 80 ký tự. Mỗi chuỗi là dãy các chữ số và các mẫu tự trong bảng mẫu tự chữ cái. Mỗi chuỗi chiếm giữ một dòng. Hãy viết một chương trình C++ để đọc các chuỗi từ file. "C:/temp/sequence.dat" và đếm số các mẫu tự và chữ số biểu thị trong mỗi chuỗi. Hãy viết kết quả xuất của chương trình này vào file "c:/temp/count.dat".*

7.10 *Hãy viết một thủ tục C++ để đọc một tên học viên và bậc lớp của học sinh đó từ file C:\exams.dat. Hãy viết tên của sinh viên và điểm trung bình và một file xuất C:\grades.dat. Sử dụng dữ liệu mẫu nhập và các chỉ dẫn xuất được chỉ định dưới đây:*

7.11 *Hãy viết các mục chương trình C++ để đọc một chuỗi dữ liệu từ một file nhập và thay thế mỗi một nguyên âm trong chuỗi với một vài ký tự khác theo các quy tắc thay thế được chỉ định như dưới đây.*

7.12 *Hãy viết một chương trình C++ để đọc một dòng từ một file nhập rồi viết dòng đó trở ngược lại thành một file xuất bảo đảm rằng dữ liệu nhập và dòng "hướng ngược" tương ứng được minh họa trong file xuất.*

7.13 *Hãy viết một chương trình C++ để đọc một dòng chuỗi từ một file và xác định xem thử dòng nhập này có phải là một palindrome hay không. Giả sử rằng mỗi một chuỗi là một dãy các ký tự mà không có dấu trống nào sử dụng dữ liệu mẫu sau đây và bảo đảm rằng các ký tự nhập tất cả đều có cùng kiểu chữ (hoa hoặc thường) có nghĩa rằng hoặc tất cả chữ hoa hoặc tất cả chữ thường.*

Dữ liệu mẫu:

RADAR
AREPERA
11111111
11113333

7.14 *Có cần thiết để kiểm tra rằng một file đã được mở đúng trước khi sử dụng nó hay không? nếu có cách nào để dễ dàng kiểm nghiệm file này được mở đúng không?*

7.15 *Hãy viết một thủ tục C++ để đọc một file nhập (C:\taxdata.dat) và hiển thị nội dung của nó. (một lần một màn hình). Giả sử rằng có 24 dòng trên màn hình.*

7.16 *Hãy viết một chương trình C để đọc một file có chứa một danh sách có các hạng mục và giá cả của nó. Có một hạng mục trên một dòng và mỗi hạng mục được theo sau là giá của nó.*

7.17 *Hãy viết một chương trình C để nhắc nhở người dùng về một tên file rồi tiến hành mở file để viết hoặc chương trình nếu có*

lỗi xuất hiện trong khi đang mở file.

7.18 Hãy viết một chương trình C để chấp nhận các dòng nhập được người dùng gõ nhập và lưu nó trong một file xuất. Chương trình hoàn tất lúc người dùng gõ nhập N cho câu hỏi "Do you want to continue"

7.19 Có điều gì sai với chương trình này?

7.20 Hãy viết một chương trình tiện ích để sao chép một file sang một file khác và trả về số các ký tự được sao chép.

7.21 Hãy viết một chương trình C để nhắc nhở người dùng về hai file, so sánh hai file này và xem thử các nội dung của chúng có giống nhau hay không?

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

7.22 Assume that we have saved a Visual Basic program in the temp directory of the C drive. Is it appropriate to refer to this program as a file?

7.23 What would happen in Visual Basic if you try to write to a file that was opened for input? Can this error be trapped and handled by the user?

7.24 What would happen if a file that is meant to be opened as an input file is opened as an output file? Is it possible to recover from this error?

7.25 Modify the procedure of Solved Problem 7.6 to keep track of the annual salary paid to the employees of the different departments. In addition, write to a new file all the information present in the input file of any employee not working for the marketing, accounting, or sales department.

7.26 What type of file is it necessary to use to read or write ASCII text files in Visual Basic?

7.27 What does the following section of code do?

```
iFileNumber = FreeFile
Open "C:\Data.dat" for Input as #iFileNumber
txtInputBox.Text = Input (LOF(iFileNumber), iFileNumber)
```

7.28 What does the following section of code do? What happens if the file is currently empty?

```
iFileNumber = FreeFile
Open "C:\Data\Text.txt" for Append as #iFileNumber
Write #iFileNumber, txtInputBox.Text
```

7.29 Assume that we have a Random file and that we want to read the entire content of file. Can we use a For Next loop to read the entire file one record at a time?

7.30 Modify the program of Solved Problem 7.8 to eliminate the leading blanks of each input string. Leading blanks are the blanks that appear before the strings.

7.31 Modify the program of Solved Problem 7.10 so that, in addition to the average grade, the student also receives a letter grade. Use the following scale for assigning grades.

Grade	Letter Grade
90–100	A
80–89	B
70–79	C
60–69	D
0–59	F

7.32 Modify Solved Problem 7.9 so that it skips and does not print any input blank to the output file.

7.33 Modify the printbackward procedure of Solved Problem 7.12 so that there are no leading blanks in any of the lines of the output file. That is, do not print any trailing blanks of the input lines.

7.34 Modify the procedure palindrome of Solved Problem 7.13 to eliminate the restriction that all input characters need to be of the same case.

7.35 In C, to test if a file has been successfully opened, we have used instructions like the ones shown below. Can these instructions be combined within a single if statement?

```
inFile=fopen("C:\temp\fileToOpen.dat","r");
if (inFile==NULL)
{
   printf("\nProgram failed to open the input file");
   exit(1);
}
```


7.36 In Solved Problem 7.18, the user was prompted to check if he wanted to continue inputting lines to the output file. The answer typed by the user was converted to an uppercase to account for the possibility of the user typing "n" instead of "N". Can this possibility be accounted for without having to convert the user response to an uppercase? Can you modify the if statement to take care of the new line character?

HƯỚNG DẪN ĐỌC HIỂU CÁC BÀI TẬP BỔ SUNG

7.22 Giả sử rằng chúng ta đã lưu một chương trình Visual Basic trong thư mục temp của ổ đĩa C. Việc tham chiếu chương trình này dưới dạng một file có phù hợp hay không?

7.23 Điều gì xảy ra trong Visual Basic nếu bạn thử viết vào một file đã được mở để nhập? Có thể lỗi này được người dùng bẫy và xử lý.

7.24 Điều gì xảy ra nếu một file có ý phải được chọn làm một file nhập thì nó lại được mở dưới dạng một file xuất? Có thể phục hồi lỗi này được không?

7.25 Hãy chỉnh sửa thủ tục ở bài tập có lời giải 7.6 để theo dõi lương hàng năm được chi trả cho các nhân viên ở các phòng ban khác nhau bên cạnh đó hãy viết một file mới dành cho tất cả thông tin hiện có trong file nhau của bất kỳ nhân viên nào không làm việc ở các phòng tiếp thị, phòng tài chánh và phòng kinh doanh.

7.26 Kiểu file nào cần được dùng để đọc hoặc viết các file text ASCII trong Visual Basic?

7.27 Một mã sau đây thực hiện điều gì?

```
iFileNumber = FreeFile
Open "C:\Data.dat" for Input as #iFileNumber
txtInputBox.Text = Input (LOF(iFileNumber), iFileNumber)
```


7.28 Một mã sau đây thực hiện điều gì? Điều gì xảy ra nếu file này hiện đang trống.

```
iFileNumber = FreeFile
Open "C:\Data\Text.txt" for Append as #iFileNumber
Write #iFileNumber, txtInputBox.Text
```

7.29 Giả sử rằng chúng ta đang có một file Random và bạn muốn đọc toàn bộ nội dung của file chúng ta có thể sử dụng vòng lập For Next để đọc toàn bộ file trên record tại một thời điểm được không?

7.30 Hãy chỉnh sửa chương trình của bài tập có lời giải 7.8 để loại bỏ các khoảng trống đứng đầu ở mỗi chuỗi nhập. Các khoảng trống đứng đầu là những khoảng trống xuất hiện trước các chuỗi.

7.31 Hãy chỉnh sửa chương trình của bài tập có lời giải 7.10 để bên cạnh tính điểm trung bình, học viên cũng nhận được một đánh giá dưới dạng mẫu tự. Sử dụng tham chiếu sau đây để gán các định mức.

7.32 Hãy chỉnh sửa chương trình của bài tập có lời giải 7.9 để nó

lướt không in bất cứ khoảng trống nhập nào sang file xuất.

7.33. Hãy chỉnh sửa thủ tục in ngược (printbackward) của bài tập cỡ lời giải 7.12 để không có khoảng trống đứng đầu nào ở bất cứ dòng của file xuất, có nghĩa rằng không được in bất cứ khoảng trống đuôi nào của các dòng nhập.

7.34 Hãy chỉnh sửa cấu trúc về thủ tục của bài tập có lời giải 7.13 để loại bỏ những giới hạn qua đó tất cả các ký tự nhập có cùng kiểu chữ (in hoa hoặc thường).

7.35 Trong C hãy thử nghiệm xem thử một file có được mở thành công hay không, chúng ta phải sử dụng các lệnh y hệt như những lệnh minh họa dưới đây. Những lệnh này có thể được kết hợp bên trong một câu lệnh ít đơn giản hay không?

7.36 Trong bài tập 7.18 người dùng được nhắc nhở để kiểm tra xem thử họ có muốn tiếp tục nhập các dòng sang các file xuất hay không? câu trả lời được người dùng gõ nhập sẽ biến đổi sang một kiểu chữ in hoa để tính đến khả năng người dùng gõ nhập "n" thay vì "N". Khả năng này có thể được tính đến mà không cần yêu cầu người dùng phải biến đổi sang một kiểu chữ in hoa được không? Bạn có thể chỉnh sửa câu đánh if để theo dõi ký tự dòng mới hay không?

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

7.22 Yes, we can say that a saved program located in the hard disk is a file. A computer science purist may argue that a program is not the same as a file since the program is contained in the file. However, for all practical purposes, we can say: "That file is a C program." This is particularly true when looking at files with any file management tool.

7.23 Visual Basic will report a "bad file mode" run-time error. Yes! This error can be trapped and handled by the user. Its number is error 54.

7.24 The contents of the file are erased. No, it is not possible to recover from this error unless there is a copy of the file saved somewhere else.

7.25 Declare four additional global variables to keep track of the department salaries. Initialize these variables to zero and modify each of the different cases of the select statements to include an instruction like cAccoutingSalaries = cSalary + cAccoutingSalaries. In addition, add an Else condition to the Select statement to account for all employees who are not part of the accounting, sales, or marketing departments. Whenever this Else condition is satisfied, write to the new file.

7.26 Sequential files are used to read or write ASCII text files.

7.27 This instruction reads the entire content of the file into the textbox txtInputBox. The Input function takes two parameters: The first parameter, LOC(iFileNumber), indicates the number of bytes that need to be read from the file. The second parameter indicates the file number from which the values will be read. Since the function returns (in bytes) the total length of the file, all characters in the file will be read in.

7.28 This code appends the text property of a textbox called txtInputBox to the file C:\Data\Text.txt. If this file is currently empty, the text will be written at the start of the file. If the file is not empty, the text will be added to the end of the file.

7.29 Yes! We can use a For Next loop to read the entire file. Calculate the highest record number in the file (see Solved Problem 7.5) and use this value as the upper limit of the For Next loop. The starting value of the loop index should be 1.

7.30 Replace the Trim() function with the Ltrim() function. This function deletes any sequence of blanks to the left of the string.

For instance, C++ does not impose any structure on files. Therefore, the notion of a "record" does not exist in C++. In C++, a file is nothing more than a sequence of bytes that ends in a special character **called the end-of-file** marker or after a specific number of bytes recorded in an administrative data structure.

7.31 The modified procedure may look like this. We have added one character variable to hold the grade assigned to the student.

```
while(!InFile.eof())
{
 InFile>>firstname>>lastname;
 sum=0;
    for(i=0; i<totalexams; i++)
    {
     InFile>>exams[i];
     sum=sum+exams[i];
    }
    average=sum/totalexams;
    if (average>=90 && average<=100)
        lettergrade='A';
    if (average>=80 && average<=89)
        lettergrade='B';
    if (average>=70 && average<=79)
        lettergrade='C';
    if (average>=60 && average<=69)
        lettergrade='D';
    if (average>=0 && average<=59)
        lettergrade='F';
    OutFile<<firstname<<' '<<lastname <<' '<<average;
    OutFile<<' '<<lettergrade<<endl;
}
}//end of calculategrades
```

7.32 Add a function that tests for blank lines. If there is a blank line do not do anything and continue reading the input file. The code for this new procedure may look like the one shown below. The code for the blankline function is not shown here.

```
void convertdata(void)
{ int i;
  while(!InFile.eof())
  {
    InFile.getline(line,sizeof(line)-1,'\n');
    if (blankline ()==0) //testing if there is a blank line
      {
```

```
        OutFile<<"In: "<<line<<endl;
        OutFile<<"Out:";
          for(i=0; i<maxlength; i++)
            {
             switch (line[i])
             {
               case 'a': //upper or lowercase a
               case 'A':
               line[i]='?';
               break;
               case 'e': //upper or lowercase e
               case 'E':
               line[i]='*';
               break;
               case 'i': //upper or lowercase i
               case 'I':
               line[i]='#';
               break;
               case 'o': //upper or lowercase o
               case 'O':
               line[i]='%';
               break;
               case 'u': //upper or lowercase u
               case 'U':
               line[i]='^';
               break;
             }// end switch
      OutFile<<line[i];
              }//end for
      OutFile<<endl;
        clearline();
        }//end of if
    }// end of while
  } //end of convertdata
```

7.33 To avoid printing any trailing blank of the input line, examine the characters of the input line beginning at the last character of the line. If the character is a blank character, ignore it until a nonblank character is encountered. Keep track of the position of this nonblank character and use it as the initial value of the loop that writes the characters to the output file. A modified printbackward procedure may look like this:

```
void printbackward(void)
{ int i,j;
  while(!InFile.eof())
  { InFile.getline(line,sizeof(line),'\n');
    OutFile<<"In: "<<line<<endl;
    OutFile<<"Out:";
        for(i=maxlength-1; i>=0 ; i--)   //find first nonblank character
          {                              //starting from the last position
           if( line[i] != ' ' )
             { j=i;
               break;
             }
          }//end for
        for(i=j-1; i>=0 ; i --) // write characters to output file
          { OutFile<<line[i];
          }//end for
          OutFile<<endl;
          clearline();
      }//end of while
} //end of printbackward
```

7.34 Only one line of the procedure needs to be modified to eliminate the restriction that all input characters have to be of the same case. The trick here is to transform the input characters, within the program, to the same case before they get to be compared. That is, we can change the case of each pair of characters to all upper or lower before we get to compare them. In the following section of code we change all input characters to uppercase using the toupper(character) function. We could have also used the tolower(character) function. In both cases, make sure that the header <ctype.h> is included at the top of the program.

```
if (toupper(line[i]) != toupper(line[j-i])) //set the flag k
   if the pair does not match
{ k=1;
   break;
}
```

7.35 Both statements can be combined in a single if statement as follows:

```
if (( inFile=fopen("C:\temp\fileToOpen.dat","r"))==NULL)
  {
    printf("\nProgram failed to open the input file");
    exit(1);
  }
```

7.36 Yes! it is possible to account for upper- or lowercase "n" simply by changing the condition within the if statement. The code to accomplish this may look like that shown below. At the same time it is possible to account for the new line character using the getchar() function again and moving it to be the "body" of the else part of the if statement. Notice the use of the getchar() function to "consume" the new line character.

```
if(answer == 'N' || answer == 'n') // exit if user does not want to continue
  {
    printf("Program ended at user request\n");
    fclose(OutFile);
    exit(0);
  }
else
  getchar();
```

CHAPTER 8

Object-Oriented Programming

Lập trình hướng đối tượng

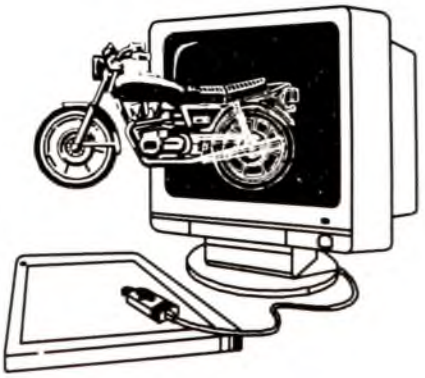
MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các khái niệm về lập trình hướng đối tượng, sự kế thừa dữ liệu, môi trường hướng đối tượng, ... Các nội dung cụ thể như sau:

- Introduction to object oriented programming
- Inheritance and data abstraction
- Advantages of object oriented programming
- Object oriented environment in Visual Basic
- Classes and inheritance in C++
- Classes and inheritance in Java

- *Giới thiệu về lập trình hướng đối tượng*
- *Sự kế thừa và sự trừu tượng của dữ liệu*
- *Các ưu điểm của việc lập trình hướng đối tượng*
- *Môi trường hướng đối tượng trong Visual Basic*
- *Lớp và sự thừa kế trong C++*
- *Lớp và sự thừa kế trong Java*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.



CHỦ ĐIỂM 8.1

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING
Giới thiệu về lập trình hướng đối tượng

The programming concepts throughout this book have all been based on traditional ***procedural programming.*** That is, through the use of procedures and functions the program obtains its input, processes that input in a specific way, and then produces the required output. The order in which instructions are executed is predetermined by the programmer, so that you can easily identify which procedures or parts of procedures are performing each task.

There are at least three potential problems with procedural programming.

It is up to the programmer to handle all possible input and output. In every program, the programmer must try to predict what mistakes a user might make and then write code to handle this input. Each type of output must be formatted carefully.

Debugging can often be difficult. Making one change in any line can produce a series of errors elsewhere that must be hunted down and fixed.

It may he difficult to reuse procedures in other code, since they were usually created and formatted for a specific program.

Over the last several years, computer scientists have been focusing their attention on ***objectoriented programming (OOP),*** or creating objects with attributes and behaviors instead of writing procedures. OOP is one way of addressing some of the problems of procedural programming. There are three main definitions needed to understand this kind of programming: objects, classes, and inheritance.

8.1.1 Object - *Đối tượng*

An object is an abstract definition of something, identified by two subparts inextricably tied together. First, an object has attributes, sometimes called ***states*** or ***properties,*** which describe the object. Second, an object has behaviors, sometimes called ***functions,*** or ***methods,*** which describe what the object can do. For example, a cat, a dog, a car, a couch, and a watermelon are all objects. Some possible attributes and behaviors for each object are listed in Table 8-1. The attributes all describe the object itself. and the behaviors describe what it can do.

Table 8-1 Attributes and Behaviors of Objects.

Object	Possible Attributes	Possible Behaviors
A cat	name, size, color, breed	eat, meow, sleep, catch mice, purr
A dog	name, size, color, breed	eat, bark, sleep, chase cars
A car	make, model, year, color, speed	accelerate, brake, honk
A couch	fabric, color, size, style	recline, open into a bed
A watermelon	color, size, number of seeds	grow, dry up, spoil

Objects defined in this abstract way are called *abstract data types* (ADTs). There are two things to notice about ADTs. First, the list of attributes and behaviors is not exhaustive. Only selected categories are included. Second, the attributes are general categories without values. For instance, the first row is not referring to a specific cat, but to any cat in general. Any cat might have a name and color and be able to eat and sleep. To describe a specific cat, one would have to fill in the values for each attribute and behavior. A specific cat might be a large white cat named Snowball, who meows softly and purrs loudly.

An ADT can be generally illustrated as in Fig. 8-1. There are two distinct kinds of behaviors:

- Actions of the object
- Settting or returning attributes

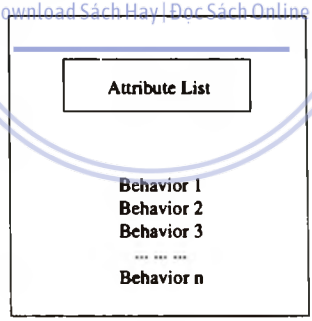


Fig. 8-1 Illustration of an object.

To protect the attributes, or variables, from corruption, such as receiving incorrect values, they are usually only accessible through the member functions. In addition to their regular actions, most classes have specific behaviors for setting and printing out the values of the member attributes. The complete collection of member functions is called the *interface*.

Usually a program contains several objects. The heart of object-oriented programming is the fact that, in contrast to procedural programming, objects usually do not do anything without being asked. *Messages*, or function calls, are sent from one object to another, invoking the member functions of the interface.

EXAMPLE 8.1 Draw a picture of a Thermostat object and a Furnace object. The Thermostat controls the Furnace.

Examine the objects in Fig. 8-2. A message comes to the Thermostat to check the current temperature. If the current setting is for heat, and the current temperature falls below the desired temperature, the Thermostat object sends a message to the Furnace object to turn on the heat.

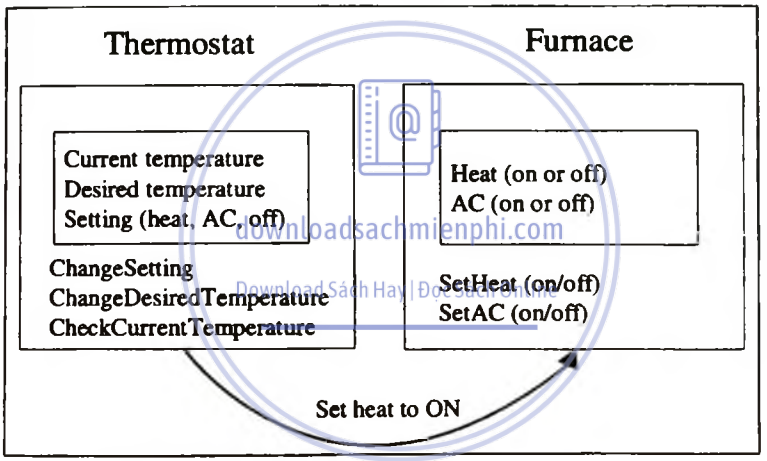


Fig. 8-2 Thermostat sending message to furnace.

As illustrated in this example, there are three parts to a message.

(1) The object to which the message is addressed (the Furnace)

(2) The function to execute (SetHeat)

(3) The parameters needed by the function (on)

Notice that the Thermostat does not directly control the private attributes of the Furnace. It merely sends a message for the Furnace to change its own settings. It is up to the Furnace to be sure that the heat and air conditioning are not operating simultaneously. See Solved Problem 8.11 for a Java implementation of these two classes.

8.1.2 Class - *Lớp*

A class is a prototype definition of an object in a specific programming language. The class definition does not include all possible attributes and behaviors, only the ones that would be needed for the specific program. For example, a program may be comparing the speed of different makes of car objects. If the color is irrelevant, that attribute does not need to be included in the definition.

Once the class is defined, the programmer must **instantiate** it, or declare one specific instance of that class. This process corresponds to declaration of built-in variable types. For example, the ADT of integer is available to the programmer. An integer in general has unique attributes (e.g., whole value, no fractional part) and behaviors (e.g., integer division). However, an integer cannot,be used until a memory location is set aside for it through a variable declaration. Only then can the location be given a value. In the same way the ADT of the class prototype is available, but cannot be used until it is instantiated and given values. Each object-oriented language handles this process differently, as shown later in this chapter.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.1

8.1 GIỚI THIỆU VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Những khái niệm lập trình hướng đối tượng trong toàn bộ cuốn sách này đặt cơ sở trên lập trình thủ tục truyền thống có nghĩa rằng thông qua phép sử dụng các thủ tục và các hàm, chương trình tìm được dữ liệu nhập của nó, xử lý dữ liệu đó theo một cách thức đặc biệt rồi tạo nên kết quả xuất cần thiết. Thứ tự mà các lĩnh vực thực thi được xác định sẵn do nhà lập trình, vì vậy bạn có thể dễ dàng nhận biết thủ tục hoặc phần thủ tục nào cần phải được thực hiện trong mỗi tác vụ.

Có ít nhất 3 khả năng với lập trình thủ tục

- *Khả năng đầu tiên là nhà lập trình phải xử lý trả dữ liệu nhập và dữ liệu xuất có thể có. Trong mỗi một chương trình nhà lập trình phải cố gắng dự đoán lỗi nào người dùng có thể mắc phải sau đó viết mã để xử lý dữ liệu nhập này. Mỗi kiểu dữ liệu xuất phải được định dạng một cách cẩn thận.*
- *Việc gỡ rối thường là một công việc khó khăn. Chỉ cần thực hiện một thay đổi trong bất kỳ dòng nào đều có thể tạo nên một chuỗi các lỗi ở bất cứ nơi nào khác và điều đó thật khó để nhận định cũng như ổn định.*
- *Khó để sử dụng lại các thủ tục trong các mã khác bởi vì chúng thường được tạo và được định dạng cho một chương trình đặc biệt.*

Trong những năm vừa qua các nhà khoa học máy tính đã tập trung chú ý về lập trình hướng đối tượng (OOP) hoặc bằng cách tạo ra các đối tượng và các thuộc tính và các đặc điểm thay vì viết các thủ tục. OOP là một cách định hướng một vài sự cố của lập trình thủ tục. Có ba định nghĩa chính cần phải hiểu kiểu lập trình này.

8.1.1 Đối tượng

Một đối tượng là một định nghĩa trừu tượng về một điều gì đó, được nhận biết do bởi hai phần gắn chặt với nhau. Trước tiên một đối tượng phải có các thuộc tính còn được gọi là các trạng thái hoặc các tính chất, hai thuộc tính này mô tả đối tượng. Thứ hai một đối tượng phải có những đặc điểm đôi khi còn được gọi là các hàm hoặc các phương pháp nhằm mô tả những gì mà đối tượng đã thực hiện, ví dụ một con mèo, một con chó và một chiếc xe hơi, v.v... tất cả đều là các đối tượng. Một vài thuộc tính và đặc điểm có thể có các đối tượng có thể liệt kê trong bảng 8.1. các thuộc tính này mô tả chính các đối tượng đó còn các đặc điểm thì mô tả những gì mà nó thực hiện.

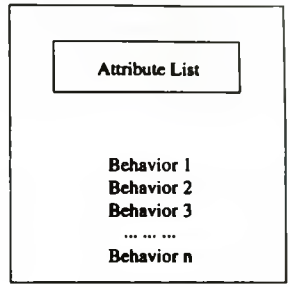
***Bảng 8.1** Các thuộc tính và đặc điểm của các đối tượng*

Object	Possible Attributes	Possible Behaviors
A cat	name, size, color, breed	eat, meow, sleep, catch mice, purr
A dog	name, size, color, breed	eat, bark, sleep, chase cars
A car	make, model, year, color, speed	accelerate, brake, honk
A couch	fabric, color, size, style	recline, open into a bed
A watermelon	color, size, number of seeds	grow, dry up, spoil

Các đối tượng được định nghĩa theo phương pháp trừu tượng này được gọi là các kiểu dữ liệu trừu tượng (ADT). Có hai điều cần lưu ý về các ADT. Trước tiên danh sách các thuộc tính, các đặc điểm không hiển nhiên. Chỉ có các hạng mục được chọn mới được đưa vào. Thứ hai các thuộc tính thường là các hạng mục tổng quát mà không có các giá trị. Điều đó có nghĩa rằng trong bảng trên đây hàng đầu tiên không tham chiếu đến một con mèo đặc biệt nhưng đến bất cứ con mèo nào nói chung. Bất cứ con mèo nào đều có một tên và màu sắc và để có khả năng ăn và ngủ, để mô tả một con mèo đặc biệt người ta phải điền vào các giá trị ứng với mỗi thuộc tính và đặc điểm. Một con mèo đặc biệt có thể là một con mèo trắng có tên là Snowball, mà tiếng kêu của nó hơi nhỏ nhẹ.

Một ADT có thể được minh họa tổng quát trong hình 8-1. Có hai loại đặc điểm phân biệt:

- *Các hành động của đối tượng*
- *Xác lập hoặc trả về các thuộc tính*

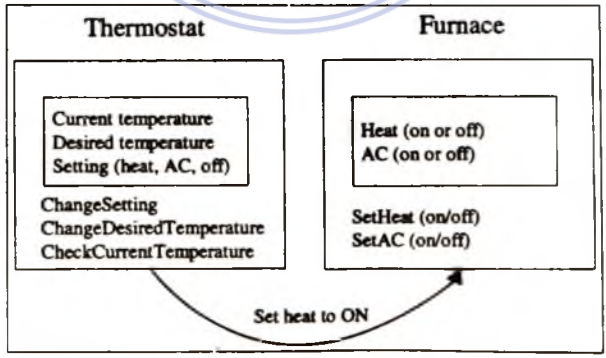


***Hình 8-1**. Minh họa một đối tượng*

Để bảo vệ các thuộc tính hoặc các biến không bị ngắt quảng, chẳng hạn như nhận các giá trị không đúng, chúng thường chỉ có thể truy cập thông qua các hàm số thành viên. Bên cạnh các hoạt động bình thường của nó hầu hết các lớp đều có các đặc điểm chuyên biệt để xác lập và in ra các giá trị của các thuộc tính thành viên. Tổng hợp hoàn chỉnh các hàm thành viên được gọi là giao diện (interface).

Thông thường một chương trình có chứa nhiều đối tượng. Trọng tâm chính của ngôn ngữ lập trình hướng đối tượng đó là ngược lại với lập trình thủ tục, các đối tượng thường không thực hiện bất cứ điều gì mà không được yêu cầu. Các thông điệp hoặc các cuộc gọi hàm được gửi từ một đối tượng này sang một đối tượng khác, viện dẫn các hàm thành viên của giao diện.

***VÍ DỤ 8.1** Vẽ một hình ảnh về đối tượng Thermostat và một đối tượng Furnace. Thermostat điều khiển Furnace.*



***Hình 8-2**. Thermostat sẽ gửi thông điệp đến Furnace*

Xem xét đối tượng trong hình 8-2. Một thông điệp đến Thermostat để kiểm tra nhiệt độ hiện tại. Nếu xác lập hiện tại là dành cho nhiệt (heat) và nhiệt độ hiện hành nằm bên dưới nhiệt độ mong muốn thì đối tượng Thermostat sẽ gửi một thông điệp đến đối tượng Furnace để mở nhiệt.

Như minh họa trong ví dụ này có ba phần của một thông điệp

(1) Đối tượng qua đó thông điệp được định hướng (Furnace) (bếp lò)

(2) Hàm để thực thi (SetHeat)

(3) Các tham số mà hàm này cần (on)

Lưu ý rằng Thermostat không trực tiếp điều khiển các thuộc tính riêng tư của Furnace. Nó chỉ gửi một thông điệp dành cho Furnace để thay đổi các cài đặt của riêng nó. Đã đến lúc Furnace phải bảo đảm rằng nhiệt và điều hòa không khí không hoạt động cùng một lúc. Xem bài tập có lời giải 8.11 dành cho việc thực thi chương trình Java của hai lớp này.

8.1.2 Lớp

Một lớp là một định nghĩa của một đối tượng theo một ngôn ngữ lập trình chuyên biệt. Định nghĩa lớp không bao gồm tất cả các thuộc tính và đặc điểm có thể có, chỉ có những đặc điểm nào và thuộc tính nào cần thiết dành cho chương trình đặc biệt mà thôi. Ví dụ một chương trình có thể so sánh sự khác biệt về tốc độ làm nên đối tượng xe hơi. Nếu màu sắc không hiển thị, thì thuộc tính đó không cần phải đưa vào trong định nghĩa.

Một khi lớp đã được định nghĩa, nhà lập trình phải thừa kế nó. (instantiate) hoặc khai báo một trường hợp đặc biệt của lớp đó. quy trình này tương ứng với việc khai báo các kiểu biến tạo sẵn. Ví dụ ADT của số nguyên luôn có sẵn cho nhà lập trình. Một số nguyên ở dạng tổng quát thì chỉ có các thuộc tính duy nhất (tức là giá trị nguyên chứ không có phần thập phân) và các đặc điểm (tức là phép chia số nguyên) tuy nhiên một số nguyên không thể được dùng cho đến khi vị trí bộ nhớ được xác lập dành cho nó thông qua việc khai báo biến. Chỉ sau đó vị trí mới được cung cấp một giá trị. Theo cách đó thì ADT của giao thức lớp là có sẵn nhưng không thể được dùng cho đến khi nó được chấp nhận và có các giá trị đã cho. Mỗi ngôn ngữ hướng đối tượng đều xử lý quy trình này theo một cách khác nhau như minh họa ở phần sau trong chương này.

CHỦ ĐIỂM 8.2

INHERITANCE AND DATA ABSTRACTION
Sự kế thừa và sự trừu tượng của dữ liệu

Classes that can be defined in terms of other classes are called ***subclasses.*** The superclass is the defining class.

EXAMPLE 8.2 Define a *car* and a truck as subclasses of the parent or superclass *motor vehicle.*

In Fig. 8-3 inheritance occurs when the specific subclass of a more general superclass has access to variables and functions from the parent superclass. All motor vehicles have wheels and headlights. The car and the truck, and even a motorcycle, do not need to repeat these attributes in their individual definitions. They can inherit the attributes from the parent class.

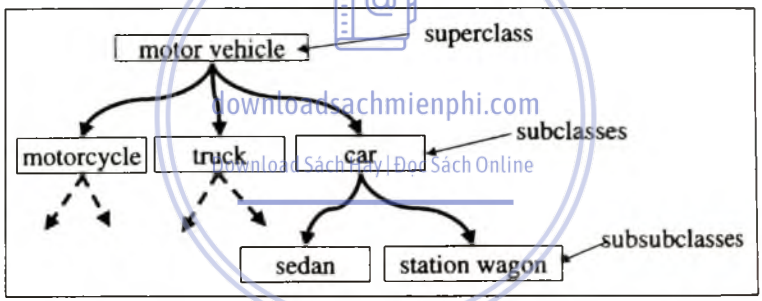


Fig. 8-3 Superclass and subdass.

In many languages a subclass can also be a superclass of another class. As shown above, car would be the superclass for the subclasses sedan and station wagon. All cars have four wheels, but only station wagons have a rear tailgate. Some languages also allow for a subclass to inherit attributes and methods from two different parents. Multiple inheritance is both complex and language specific, and thus will not be addressed in this book.

Superclasses are general, subclasses are more specific. The subclass inherits everything from its superclass, and does not need to redefine the same attributes. The subclass definition only contains what is unique to it.

Some programming languages allow for a superclass to be an abstract class with possibly undefined methods. Each subclass can define the method as needed. This is called polymorphism and is beyond the scope of this book.

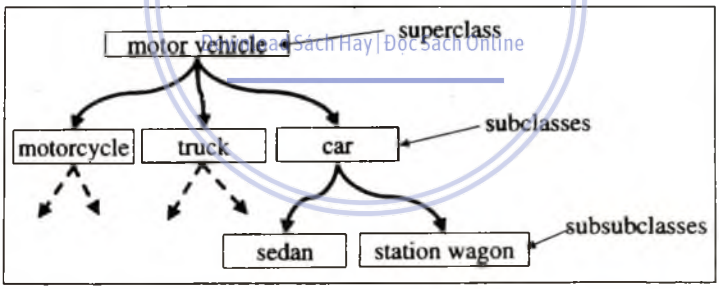
One of the most important goals of object-oriented programming ***is data abstraction,*** or keeping WHAT an object can do separate from HOW it does it. The interface of the class (the group of member functions) gives the other objects and programs information about what it can do, not how. The thermostat doesn't need to know how the furnace actually works, just how to turn it on. A list object might have a sorting function, but the program calling that function does not need to know whether it is a bubble sort or a selection sort.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.2

8.2 TÍNH THỪA KẾ VÀ TÍNH TRỪU TƯỢNG CỦA DỮ LIỆU

Các lớp có thể được định nghĩa theo các thuật ngữ của các lớp khác được gọi là lớp con. Các siêu lớp chính là phần định nghĩa lớp.

VÍ DỤ 8.2 *Xác định một chiếc xe hơi và một chiếc xe tải làm các lớp con của lớp bố hoặc siêu lớp motor vehicle. Trong hình 8-3 tính thừa kế xảy ra lúc các lớp con đặc biệt của siêu lớp tổng quát hơn, truy cập vào các biến và các hàm từ các siêu lớp bố. Tất cả các xe hơi và xe tải đều có 4 bánh và có đèn trước, xe hơi và xe tải thậm chí xe mô tô đều không cần phải lập lại các thuộc tính này trong phần chuyên biệt của chúng. Chúng có thể thừa kế các thuộc tính từ các lớp bố.*



Hình 8-3*. Các siêu lớp và các lớp con*

Trong nhiều ngôn ngữ, một lớp con cũng có thể là một siêu lớp của một lớp khác như minh họa trên đây, chiếc xe hơi sẽ là siêu lớp đành cho các lớp con sedan và station wagon. Tất cả các xe hơi đều có nhưng chỉ có các toa xe mới có các đuôi gắm đằng sau. Một vài ngôn ngữ cũng cho phép một lớp con thừa kế các thuộc tính và các phương pháp từ hai bố khác nhau. Tính đa thừa kế là một đặc điểm phức tạp và mang tính ngôn ngữ vì vậy nó sẽ không được đề cập trong sách này.

Các siêu lớp thì mang tính tổng quát còn các lớp con thì mang tính chuyên biệt. Các lớp con thừa kế mọi thứ từ các siêu lớp và nó không cần phải xác định lại cùng các thuộc tính giống nhau. Định nghĩa lớp con chỉ có chứa những gì đơn nhất đối với nó mà thôi. Một vài ngôn ngữ lập trình cho phép một siêu lớp phải là một lớp trừu tượng và các phương pháp không được xác định. Mỗi lớp con có thể xác định phương pháp nếu cần. Đây được gọi là tính đa hình thái nó vượt xa khuôn khổ của quyển sách này.

Một trong những mục tiêu quan trọng nhất của lập trình hướng đối tượng đó là tính trừu tượng của dữ liệu hoặc theo dõi những gì mà một đối tượng có thể tách rời với cách mà nó thực hiện. Giao diện của lớp này (các hàm thành viên) cho các đối tượng khác và các thông tin chương trình biết về những gì mà nó thực hiện chứ không trình bày cách thực hiện của nó. Thermostat không cần phải biết cách hoạt động của lò hơi, nó chỉ biết cách mở lò hoạt động mà thôi. Một đối tượng danh sách có thể có một hàm phân loại nhưng chương trình để gọi hàm đó thì không cần phải biết phép phân loại bọt hay phép phân loại chọn lựa.

CHỦ ĐIỂM 8.3

ADVANTAGES OF OBJECT ORIENTED PROGRAMMING

Các ưu điểm của việc lập trình hướng đối tượng

The advantages of object-oriented programming are easy to identify by going back to the problems of procedural programming explained at the beginning of this chapter. OOP provides a direct response to each problem. Table 8-2 describes the advantages of object-oriented programming over procedural programming.

Table 8-2 Advantages of OOP

Procedural Programming	Object-Oriented Programming
The programmer must handle all possible input and output, writing code to handle all input and formatting all output.	The object is responsible for its own input and output, protecting the variables from invalid values and appropriately formatting all output.
Debugging can often be difficult, especially finding errors because of changes to code.	Debugging is much easier. Since the object is programmed to behave in a specific fashion, it is easy to pinpoint errors.
It is difficult to reuse procedures in other code, since they are usually program specific.	It is easy to reuse code. Once an object is defined, it can be used in any program.
Data abstraction is hard to implement.	Data abstraction is easy to implement.

In summary, object-oriented programming provides an environment which optimizes reuse of code, and allows for future changes.

EXAMPLE 8.3 Explain how a circle class could be created and reused.

The programmer creates a circle class. The circle knows its own location, how to set its radius, and how to draw itself Any program needing a circle can easily import this class definition and use it without rewriting any other code. Also, a class definition of a superclass *shape* with subclasses of *circle* and *rectangle* can easily be extended by adding a subclass for *triangle* or *trapezoid*.

The next three sections will describe some of the object-oriented features and the use of classes in Visual Basic, C++, and Java. C is an older language that has no object-oriented features, so it will not be addressed.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.3

8.3 ƯU ĐIỂM CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Ưu điểm của lập trình hướng đối tượng thật dễ dàng nhận biết khi bạn trở lại với những vấn đề về lập trình thủ tục đã được giải thích ở phần đầu chương này. OOP cung cấp một sự đáp ứng trực tiếp đối với mỗi vấn đề. Bảng 8-2 mô tả ưu điểm của lập trình hướng đối tượng so với lập trình thủ tục.

***Bảng 8-2** Ưu điểm của OOP*

Lập trình thủ tục	*Lập trình hướng đối tượng*
Nhà lập trình phải xử lý tất cả các dữ liệu nhập và xuất có thể có, viết mã để xử lý tất cả các dữ liệu nhập và định dạng tất cả các dữ liệu xuất.	*Đối tượng có trách nhiệm với dữ liệu nhập và xuất của riêng nó, bảo vệ các biến khỏi gặp những giá trị không đúng và định dạng phù hợp tất cả các kết quả xuất.*
Việc gỡ rối có thể khó khăn đặc biệt việc tìm các lỗi do bởi sự thay đổi của mã.	*Việc gỡ rối dễ dàng hơn nhiều. Bởi vì đối tượng được lập trình theo một cách thức chuyên biệt cho nên dễ dàng nhận định ra các lỗi.*
Khó để sử dụng lại các thủ tục trong màng khác, bởi vì chúng thường là chương trình chuyên biệt.	*Dễ sử dụng lại mã, một khi đối tượng được xác định nó có thể được dùng trong bất cứ chương trình nào.*
Tính trừu tượng của dữ liệu thật khó thực thi	*Tính trừu tượng của dữ liệu dễ thực thi*

Tóm lại, lập trình hướng đối tượng cung cấp một môi trường qua đó tối ưu hóa việc sử dụng mã và cho phép có các thay đổi trong tương lai.

***VÍ DỤ 8.3** Giải thích tất cả cách một lớp vòng tròn có thể được tạo và được sử dụng lại. Người lập trình phải tạo một lớp hình tròn, hình tròn biết được vị trí của riêng nó, cách xác lập bán kính và cách vẽ. Bất cứ chương trình nào cần một hình tròn đều có thể dễ dàng nhập định nghĩa lớp này và sử dụng nó mà không cần viết lại bất cứ mã nào khác. Cũng vậy một định nghĩa lớp thuộc một hình dạng siêu lớp với các lớp con là hình tròn và hình chữ nhật có thể dễ dàng được mở rộng bằng cách bổ sung thêm lớp con dành cho tam giác hoặc hình than.*

Ba phần kế tiếp dưới đây sẽ mô tả một vài đặc tính hướng đối tượng và việc sử dụng các lớp trong Visual Basic, C++ và Java. C là một ngôn ngữ cũ hơn nó không có các đặc tính hướng đối tượng vì vậy chúng ta sẽ không đề cập nó ở đây.

CHỦ ĐIỂM 8.4

OBJECT ORIENTED ENVIRONMENT IN VISUAL BASIC

Môi trường hướng đối tượng trong Visual Basic

Even though Visual Basic is not a true object-oriented programming language, the programmer is working in an object-oriented environment. Microsoft calls Visual Basic an ***event-driven*** programming language. In event-driven programming, the user determines the sequence of instructions to be executed, not the programmer. The program screen is built using various buttons and boxes and the user creates *events* by pressing keys or moving the mouse. The program is not a precise sequence of instructions to be executed, but a collection of procedures that contain the code needed to respond to events.

EXAMPLE 8.4 A program screen in Visual Basic might look like the one shown in Fig. 8-4. Explain some of its event-driven features.

The user enters numbers into the boxes and clicks on the command button called *Add*. The program then executes the code that was written to handle the adding process. Code is generally written for all the buttons to handle any possible choice the user might make, but no code is actually executed until the user triggers the event by clicking the button.

In Visual Basic, the program window, or **form,** and all the items placed inside it, are objects. As in OOP, each object has behaviors and attributes. The behaviors, called Subs in Visual Basic, are the code segments written to handle a variety of events. The attributes, called ***properties,*** can be set by the programmer during the design phase of programming, or may be set by code.

In Fig. 8-4, the form contains ten objects: four command buttons, three text boxes, and three labels. Text boxes are used for user input/output, and labels for explanation purposes. Command buttons can contain code to respond to events such as a mouse click, a double click, or a mouseover. All these objects have properties such as name, color, and size. Forms, labels, and buttons have captions, and textboxes of course have text. Many other control objects are also available.

In the Visual Basic environment, the programmer manipulates built-in screen objects, setting their properties and programming their behaviors. In C++ and in Java, the programmer has the ability to create classes and subclasses, to specify which attributes and behaviors will be used, and to define inheritance precisely.

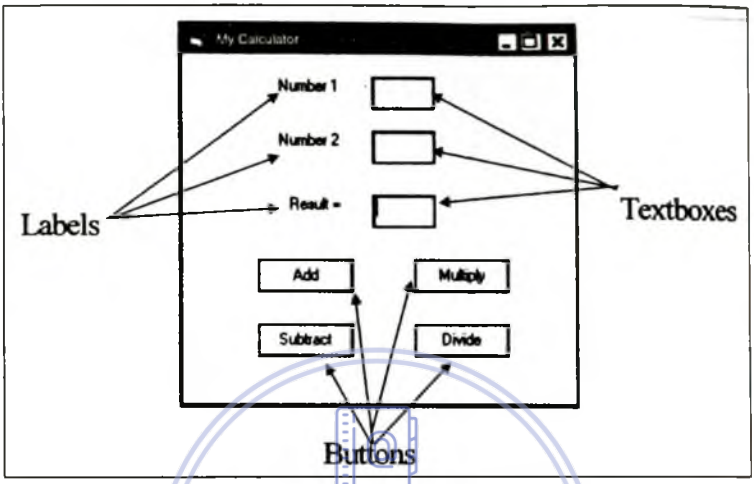


Fig. 8-4 Visual Basic calculator example

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.4

8.4 MÔI TRƯỜNG HƯỚNG ĐỐI TƯỢNG TRONG VISUAL BASIC

Thậm chí mặc dù Visual Basic không phải là một ngôn ngữ lập trình hướng đối tượng đúng, nhà lập trình vẫn đang làm việc trong môi trường hướng đối tượng. Microsoft gọi Visual Basic là một ngôn ngữ chương trình vận hành biến cố. Trong chương trình vận hành biến cố người dùng xác định chuỗi các lệnh phải được thực thi chứ không phải là người lập trình. Màn hình chương trình được tạo bằng cách sử dụng các nút khác và các ô khác nhau, người dùng tạo các biến cố bằng cách nhấn các phím hoặc di chuyển chuột. Chương trình này không phải là chuỗi các lệnh chính xác và được thực thi nhưng là một tập hợp các thủ tục có chứa mã cần thiết để đáp ứng với các biến cố.

VÍ DỤ 8.4 *Một màn hình chương trình trong Visual Basic giống như màn hình được minh họa trong hình 8-4. hãy giải thích một vài đặc tính biến cố tự vận hành.*

Người dùng nhập vào những con số trong các ô và nhích lên nút lệnh có tên là Add. Sau đó chương trình thực thi mã được viết để xử lý quy trình cộng này. Mã thường được viết dành cho tất cả các nút để xử lý

bất cứ sự chọn lựa nào có thể có do người dùng thực hiện, nhưng không có mã nào thật sự được thực thi cho đến khi người dùng kích khởi biến cố bằng cách nhấp lên nút này.

Trong Visual Basic cửa sổ chương trình hoặc form và tất cả các hạng mục đều đã đặt vào bên trong nó, gọi là các đối tượng. Cũng như trong OOP mỗi một đối tượng đều có các đặc điểm và các thuộc tính. Các đặc điểm được gọi là Subs trong Visual Basic là các đoạn mã được viết để xử lý một loạt các biến cố. Các thuộc tính được gọi là properties có thể được xác lập đo nhà lập trình trong suốt giai đoạn thiết kế chương trình hoặc có thể được xác lập theo mã.

Trong hình 8-4, form có chứa 10 đối tượng: 4 nút lệnh, 3 hộp text boxes, và 3 nhãn các text boxes được dùng để cho người dùng nhập – xuất các nhãn dành cho mục đích giải thích. Các nút lệnh có thể chứa mã để đáp ứng cácbiến cố chẳng hạn như nhấp chuột, nhấp đúp hoặc lượn chuột. Tất cả những đối tượng này đều có các đặc tính chẳng hạn như tên, màu và kích cỡ. Các dạng, nhãn và nút đều có các chú thích và các text boxes lẽ dĩ nhiên là có text. Nhiều đối tượng điều khiển khác cũng có sẵn.

Trong môi trường Visual Basic người lập trình phải xử lý các đối tượng trên màn hình được tạo sẵn, xác lập các tính chất của chúng và lập trình đặc điểm của nó. Trong C++ và trong Java người lập trình phải có khả năng tạo các lớp và các lớp con. Để chuyên biệt hóa loại thuộc tính và đặc điểm nào phải sử dụng và để định nghĩa tính thừa kế một cách chính xác.

CHỦ ĐIỂM 8.5

CLASSES AND INHERITANCE IN C++
Lớp và sự thừa kế trong C++

Classes do not exist in C. The addition of objects was one of the reasons for developing the C++ language. The C++ terminology for object-oriented programming concepts is shown in Table 8-3.

Table 8-3 Terminology for OOP in C++.

OOP	C++
Object	Class instance or class object
Instance variable or attribute	Private data member
Method	Public member function
Message passing	Function calls to public member functions

C++ is a mixture of procedural and object-oriented programming. Classes are defined and implemented. However, there must also be a main() procedure to start the program and initiate the function calls.

In C++, the definition of the class includes the definitions for both the member variables (attributes) and the member functions (behaviors). To provide for as much data abstraction as possible, the declaration of the class is in a different section, or even a different file, from the implementation, or. code, for the member functions. The general format of a class definition is shown in Fig. 8-5.

```
//class definition (located in classname.h file)
class classname
{
   member variables list
   member function list (usually prototypes, or headings only)
}; //notice semicolon after closing brace
-----------------------------------------------------------------
//class implementation (located in classname.cpp file)
complete code for each function
```

Fig. 8-5 Format for loss definition.

8.5.1 Class Header File (.h)

The code where the class object is defined is usually contained within a file with the extension .h, which stands for header.

EXAMPLE 8.5 Write C++ definition of a class Light. The light maybe on or off: A program must be able to print out the current status of the light and to flip the light switch. For a specific example of this C++ class, look at Fig. 8-6.

```
//class definition (Light.h file)
class Light
{
   private:
      int status;              //Light is on(1) or off(0)
   public:
      Light();                 //default constructor
      Light(int statusIn);     //constructor to specific status
      int flipSwitch();        //change the status
      void printStatus();      //print the current status
};
```

Fig. 8-6 Light.h.

The header file, or file ending in .h, provides the interface for how this class will interact with the outside world. The access specifiers public and private determine specific access rights. Variables or functions defined as public can be accessed from anywhere, and those defined private are accessible only within the class itself; For security purposes, usually the variables are private, protected from change by any other section of code except by member functions. The functions are usually public so the class can be used by other code segments.

In this example, the name of the class is *Light*. The usual convention is to begin class names with a capital letter. The body of the definition, within the braces, lists the member variables and functions. The only variable is the status of the light, on or oft: Functions are defined to flip the switch (turn it off or on) and to print out the current status of the light.

Note the functions that have the same name as the class and no return type. These are ***constructors,*** functions called automatically when an instance of the ADT is declared. In this case, there are two, one without any parameters and one with an incoming integer parameter. Before examining these functions, first consider an integer. You may declare an integer with or without a value.

```
int num1;        //no initial value given
int num2=9;      //initial value of 9 given
```

Since integer is a built-in data type, the compiler knows in both of these instances to set aside a memory location for type integer, and in the second case to put in a value of 9. We are defining classes that will later be used by a program. Therefore, we must specify what to do when an object of our class is declared. In this case, we are allowing an object to be declared with or without an initial value specified. Specifying two or more functions with the same name is an example of ***overloading*** functions. When functions are overloaded, they must differ in the number and/or type of parameters. Usually there are at least two constructors, one of which is the default constructor without parameters. If an array of the class objects will be declared, there must be a default constructor.

Because we have specified both types of constructors, we will allow both kinds of ADT object declaration:

```
Light myLight; //call default constructor to set aside a
  memory location
Light yourLight(1); //set aside a memory location and set
  initial value to on
Light anotherLight(0); //set aside a memory location and set
  initial value to off
```


8.5.2 Class Implementation File (.cpp) - *File thực thi lớp*

The header file provides the interface, or WHAT the class can do. The implementation file, ending in .cpp, shows HOW each function is implemented. The member functions have access to all the member variables. However, because the implementation is in a separate file, a connection must be made between the definition and the implementation. The header file is included with the compiler directive (#) at the top. If the file were not in the same directory, the entire pathway would be enclosed in the quotation marks. The code for Light.cpp is shown in Fig. 8-7.

Member functions follow all the rules of other C++ functions, with two exceptions. First, as stated before, the constructors do not have a return type. Second, all the function names are preceded by the class name and the scope resolution operator (::) which identifies the function as a member of a specific class. Every member function implemented in the .cpp file must have its prototype, or heading, listed in the .h file.

8.5.3 Using the C++ Class - *Sử dụng lớp C++*

The header and implementation files together completely define the new ADT The class is now ready to be used by a calling program.

```
#include <iostream.h>          //library necessary to use cout
#include "Light.h"             //necessary to connect to header file

//class implementation (Light.cpp file)
Light::Light()                 //default constructor
{
   status=0;                   //0 means off, begin with it off
}

Light::Light(int statusIn)     //constructor to specific status
{
   status=statusIn;
}
int Light::flipSwitch()        //change the status
{
   status=!status;
   //if it is off, turn it on. if it is on, turn it off
   return status;
}

void Light::printStatus()      //print the current status
{
   if (status==0) cout<<"The Light is off";
   else cout<<"The Light is on";
   cout<<endl;
 }
```


Fig. 8-7 Light.cpp.

EXAMPLE 8.6 Write a simple C++ program to declare instances of the class Light and print the values, they change the values and print their values again.

The implementation of this simple program is shown in Fig. 8-8.

```
#include<iostream.h>          //necessary to use cout
#include "Light.h"            //necessary to connect to header file

void main ()
{
   Light myLight;             //invokes default constructor
   Light yourLight(1);        //invokes constructor with a parameter

   myLight.printStatus();     //prints current values, off and on
   yourLight.printStatus();

   myLight.flipSwitch();      //changes values
   yourLight.flipSwitch();

   myLight.printStatus();     //prints current values, on and off
   yourLight.printStatus();
}
```

Fig. 8-8 Using class Light.

The output from this main() function would be:

```
The Light is off
The Light is on
The Light is on
The Light is off
```

In order to declare instances of the class *Light*, the header file must be included. Then two specific instances are created. The first, *myLight*, is initially set to off by the default constructor. The second, *yourLight*, is specifically set to on when it is created. (Remember, 1 is on, 0 is off.)

The member functions are invoked through the use of the **dot operator** (.). The general form is *objectfunctionO* or *objectvariable*. In this program, it is impossible to access the status directly (as in myLight. status= 1;) because the member variable was declared private. By allowing the member variables only to be changed by member functions, the class protects itself from accidental changes or invalid data.

If the calling program wanted to declare an array of ten Lights, each item in the array is accessible through the subscript. The declaration would look like this:

```
Light myLights[10];
```


After setting some values, each individual light could be accessed and printed out using a loop:

```
for (j=0; j<10; j++) myLights[j].printStatus();
```

8.5.4 Inheritance in C++ - *Sự thừa kế trong C++*

C++ allows for inheritance, where subclasses can be derived from a superclass, as shown in Fig. 8-9. The derived classes inherit the member variables and functions from the parent class.

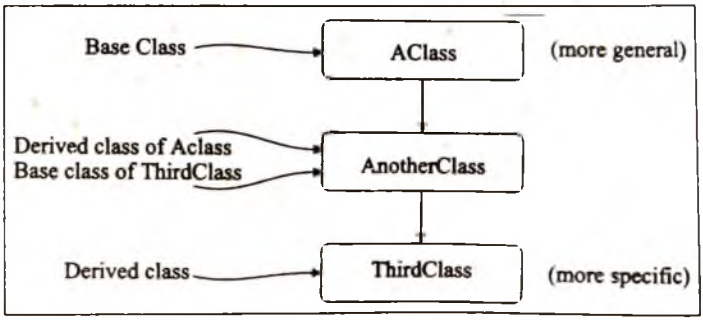


Fig. 8-9 Base and derived classes.

However, if any of the members have been declared as *private,* then even the derived class cannot i access them. For example, a car could have a member variable of color inherited from the superclass motor vehicle, but it would not be able to access that value with its member functions. If the members of the parent class were declared as public, then the derived class could access them, but so could any other code segments, and the element of protection and data hiding would be lost. Therefore, C++ allows another level of access specifier in addition to public and *private,* that of **protected.** When a member is declared as protected, it can be inherited and accessed by any subclasses, but is still inaccessible from other code segments.

EXAMPLE 8.7 Construct an ADT Jeans that is a more specific type of Clothing.

To illustrate the concept of inheritance in C++, consider Dg. 8-10. Clothing would be the superclass and Jeans would be the derived class. Possible member variables and functions are shown.

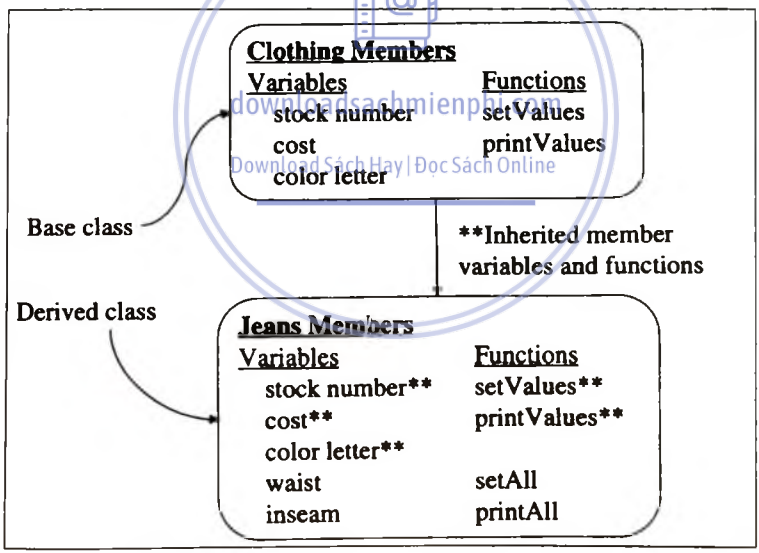


Fig. 8-10 ADT for Clothing and Jeans.

In a particular store, all the clothing would have a stock number, cost, and a color code. These variables would be inherited by any subclass derived from Clothing. Only jeans might need a waist size and inseam length.

Shirts would need a collar size and sleeve length. Other types of clothing might need additional variables.

To implement this ADT in C++, consider the code in Fig. 8-11. First the Clothing.h and Clothing.cpp files are created, and then the Jeans.h and Jeans.cpp files are written. The Clothing.h must be included in the Jeans.h file to indicate the source of the inherited members, and then specified in the first line of the class definition in this form:

```
class subclassName : classAccess parentClassName
```

Because the instance variables in the Clothing class are protected, they are accessible in the Jeans class. The Jeans::setAll(...) function could have included the code:

```
stockNum=n; cost=c; colorID=col;
```

That code is not there because it is much easier for the derived class to make use of the parent's setValues(...) function. Notice, however, that no inherited data members are set in the *Jeans* constructor. The *Clothing* constructor is automatically called first when a *Jeans* object is instantiated.

```
//Clothing.h-header for superclass for clothing

class Clothing
{
  protected:
        int stockNum;
        double cost;
        char colorID;
  public:
        Clothing();                 //constructor
        void setValues (int numIn, double costIn, char colorIn);
        void printValues();
);      double reportCost();
```

```
//Clothing.cpp-implementation of superclass

#include<iostream.h>
#include"Clothing.h"

Clothing::Clothing()                //constructor-zero values
{       stockNum=0; cost=0.0; colorID='';}

void Clothing::setValues (int numIn, double costIn, char colorIn)
//allow the setting of all Clothing values
{       stockNum=numIn; cost=costIn; colorID=colorIn; }

double Clothing::reportCost()       //report the cost
{ return cost;    }

void Clothing::printValues()        //print out the Clothing Info
{       cout<<"Item "<<stockNum<<" costs "<<cost
        <<" with color code "<<colorID;
}
```

Fig. 8-11 Clothing class and Jeans class.

```
//Jeans.h-definition for the Jeans class

#include "Clothing.h"

class Jeans:public Clothing      //indicates superclass
{
  protected:
        int waist;
        int inseam;
  public:
        Jeans();                 //constructor
        void setAll(int n, double c, char col, int w, int i);
        //set all values for Jeans
        void printAll();         //print all info
};
```

```
//Jeans.cpp-implementation of Jeans class

#include <iostream.h>
#include "Jeans/h"

Jeans::Jeans()                   //constructor
{       waist=0; inseam=0; }

void Jeans::setAll(int n, double c, char col, int w, int i)
//set all values for Jeans
{
  setValues(n, c, col);          //calls superclass function
  waist=w; inseam=i;             //sets subclass unique values
}

void Jeans::printALL()           //print unique info
{       printValues();           //calls superclass function
        cout<<" waist="<<waist<<" and inseam="<<inseam;
}
```


Fig. 8-11 *(continued)*

A short testing main() function could be written as shown in Fig. 8-12. The Clothing.h is not included in the main because it is only needed in the Jeans.h file. When the object *myPants is* instantiated, first the *Clothing* constructor is called and then the *Jeans* constructor. The output from this main() would be:

Item 55 costs 24.99 with color code b waist = 34 and inseam = 30

```
// main() to test jeans

#include <iostream.h>
#include "Jeans.h"

void main ()
{
    Jeans myPants;
    myPants.setAll(55, 24.99, 'b', 34, 30);      //sets all values
    myPants.printAll();                          //prints all
    cout<<endl;
}
```

Fig. 8-12 Testing the Jeans class.

C++ allows for base and inherited classes. However, it is not a pure object-oriented language because it still needs the main() function to issue the calls to the classes. Java is one step further toward a true object-oriented language.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.5

8.5 CÁC LỚP VÀ TÍNH THỪA KẾ TRONG C++

Các lớp không hiện diện trong C. Sự bổ sung các đối tượng là một trong những lý do để phát triển ngôn ngữ C++. Thuật ngữ C++ dành cho khái niệm lập trình đối tượng được minh họa trong bảng 8-3.

***Bảng 8-3.** Thuật ngữ dành cho OOP trong C++*

OOP	*C++*
Đối tượng	*Trường hợp lớp hoặc đối tượng lớp*
Biến trường hợp và thuộc tính	*Thành phần dữ liệu chuyên biệt*
Phương pháp	*Hàm thành phần công cộng*
Truyền thông điệp	*Hàm gọi đến các hàm thành phần công cộng*

C++ là một hỗn hợp lập trình thủ tục và lập trình hướng đối tượng. Các lớp được xác định và được thực thi. Tuy nhiên, cũng phải có một thủ tục main() để khởi động chương trình và khởi tạo các cuộc gọi hàm.

Trong C++ định nghĩa của lớp bao gồm các định nghĩa của các biến phần tử (các thuộc tính) và các hàm phần tử (đặc tính) để cung cấp càng nhiều tính trừu tượng càng tốt. Phần khai báo của lớp sẽ nằm ở một mục khác hoặc thậm chí nằm trong một file khác, khác việc tự thực thi, viết mã dành cho các hàm thành phần. Dạng tổng quát của một định nghĩa là được minh họa như trong hình 8-5.

```
//class definition (located in classname.h file)
class classname
{
   member variables list
   member function list (usually prototypes, or headings only)
}; //notice semicolon after closing brace
-----------------------------------------------------------------
//class implementation (located in classname.cpp file)
complete code for each function
```

***Hình 8-5.** Các dạng dành cho định nghĩa lớp*

8.5.1 File tiêu đề lớp (.h)

Mã ở đó đối tượng lập được định nghĩa thường có chứa bên trong một file phần mở rộng là .h, nó viết tắt của chữ header.

***VÍ DỤ 8.5** Hãy viết định nghĩa C++ dành cho một lớp Light (đèn). Đèn có thể ở trạng thái mở hoặc tắt, một chương trình có thể in ra trạng thái hiện tại của đèn và lật công tắc đèn.*

Đối với một ví dụ mẫu trong lớp C++ này hãy xem hình 8-6.

```
//class definition (Light.h file)
class Light
{
   private:
      int status;              //Light is on(1) or off(0)
   public:
      Light();                 //default constructor
      Light(int statusIn);     //constructor to specific status
      int flipSwitch();        //change the status
      void printStatus();      //print the current status
};
```

***Hình 8-6.** Light h.*

File tiêu đề hoặc phần đuôi file trong .h cung cấp giao diện về cách lớp này sẽ giao tiếp với thế giới bên ngoài. Các đặc trưng truy cập public và private xác định quyền truy cập đặc biệt. Các biến và các hàm được định nghĩa dưới dạng công cộng có thể được truy cập từ bất cứ nơi nào và các biến hoặc các hàm được định nghĩa dưới dạng riêng tư thì chỉ có thể được truy cập bên trong chính lớp đó mà thôi. Vì mục đích an toàn thường các biến là riêng tư được bảo vệ để tránh bất cứ sự thay đổi bởi bộ phận mã khác ngoại trừ các hàm thành phần. Các hàm này thường là công cộng để lớp có thể được dùng bởi các đoạn mã khác.

Trong ví dụ này tên của lớp là light. Quy ước thông thường đó là phải bắt đầu các tên lớp với một mẫu tự chữ hoa. Nội dung của phần định nghĩa được đặt bên trong các dấu móc ngoặc, liệt kê các diễn hình riêng và các hàm. Điểm duy nhất là trạng thái của đèn on hoặc off. Các hàm được xác định để bật công tắc (mở hoặc tắt nó) để in ra trạng thái hiện tại của đèn.

Lưu ý các hàm phải có tên giống hệt như tên của lớp và không có kiểu trả về. Đây là các constructors (công cụ để tạo), các hàm được gọi tự động lúc có một trường hợp ADT được khai báo. Trong trường hợp này thì có hai, trong đó một thì không có bất cứ tham số nào và một

có một tham số nguyên gửi đến, trước khi xem xét các hàm này trước tiên phải xem xét một số nguyên. Bạn có thể khai báo một số nguyên với trường hợp có hoặc không có một giá trị.

```
int num1;              //no initial value given
int num2=9;            //initial value of 9 given
```

Bởi vì integer là một kiểu dữ liệu được tạo sẵn cho nên trình biên soạn viết cả hai trường hợp này để xác lập một vị trí bộ nhớ dành để gõ nhập số nguyên và trong trường hợp thứ hai nó đưa vào một giá trị bằng 9. chúng ta đang xác định các lớp vốn sẽ được chương trình sử dụng ở phần sau trong đó chúng ta phải chỉ định những gì phải làm lúc một đối tượng của lớp được khai báo. Trong trường hợp này chúng ta cho phép một đối tượng được khai báo với trường hợp có hoặc không có giá trị khởi tạo được chỉ định. Việc chỉ định hai hoặc nhiều hàm có cùng tên là một ví dụ của các hàm quá tải (overloading). Lúc các hàm này bị quá tải, chúng phải khác nhau về kiểu số và hoặc kiểu tham số. Thông thường có ít nhất là hai bộ cấu tạo, một bộ cấu tạo mặc định không có tham số. Nếu một mảng các đối tượng lớp được khai báo, thì bắt buộc phải có một bộ cấu tạo mặc định.

Bởi vì chúng ta phải chỉ định hai kiểu bộ cấu tạo, chúng ta sẽ cho phép cả hai kiểu đối tượng ADT được khai báo.


```
Light myLight;          //call default constructor to set aside a memory location
Light yourLight(1);     //set aside a memory location and set initial value to on
Light anotherLight(0); //set aside a memory location and set initial value to off
```

8.5.2 File thực thi lớp (.cpp)

File header cung cấp giao diện, hoặc cung cấp những gì mà lớp có thể thực hiện. File thực thi kết thúc bằng .cpp cho thấy cách mà mỗi hàm được thực thi. Các hàm phần tử phải được truy cập vào các biến phần tử. Tuy nhiên, bởi vì việc thực thi phải nằm trong một file riêng biệt, cho nên phải có một nối kết giữa định nghĩa và thực thi. File header được đưa với phần chỉ dẫn trình biên soạn (#) ở đầu. Nếu file này không nằm trong cùng một thư mục, thì toàn bộ đường dẫn sẽ được đưa vào các dấu ngoặc kép. Mã dành cho light.cpp được minh họa trong hình 8-3.

Các hàm phần tử phải tuân theo tất cả các quy tắc của các hàm C++ khác ngoại trừ hai điều thứ nhất như đã phát biểu trước đây như bộ cấu tạo không có một kiểu trả về. Thứ hai tất cả các tên hàm phải có tên lớp và toán tử phân giải phạm vi (::) đứng trước chỉ định hàm như là một thành viên của một lập đặc biệt. Mỗi hàm phần tử phải được

thực thi trong file .cpp phải có giao thức của nó hoặc tiêu đề được liệt kê trong file .h.

8.5.3 Sử dụng lớp C++

Tiêu đề và file thực thi cả hai xác định hoàn chỉnh ADT mới. Lớp bây giờ chuẩn bị được dùng bằng chương trình gọi.

```
#include <iostream.h>          //library necessary to use cout
#include "Light.h"             //necessary to connect to header file

//class implementation (Light.cpp file)
Light::Light()                 //default constructor
{
   status=0;                   //0 means off, begin with it off
}
Light::Light(int statusIn)     //constructor to specific status
{
   status=statusIn;
}
int Light::flipSwitch()        //change the status
{
   status=!status;
   //if it is off, turn it on. if it is on, turn it off
   return status;
}
void Light::printStatus()      //print the current status
{
   if (status==0) cout<<"The Light is off";
   else cout<<"The Light is on";
   cout<<endl;
}
```


***Hình 8-7** Light.cpp*

***VÍ DỤ 8.6** Hãy viết một chương trình C++ để khai báo các trường hợp của lớp Light và in các giá trị rồi thay đổi các giá trị và in các giá trị của chúng một lần nữa.*

Việc thực thi chương trình đơn giản này được minh họa trong hình 8.8

```
#include<iostream.h>          //necessary to use cout
#include "Light.h"            //necessary to connect to header file

void main ()
{
   Light myLight;             //invokes default constructor
   Light yourLight(1);        //invokes constructor with a parameter

   myLight.printStatus();     //prints current values, off and on
   yourLight.printStatus();

   myLight.flipSwitch();      //changes values
   yourLight.flipSwitch();

   myLight.printStatus();     //prints current values, on and off
   yourLight.printStatus();
}
```

***Hình 8-8** Sử dụng lớp Light*

Kết quả xuất từ hàm main() này sẽ là:

The Light is off
The Light is on
The Light is on
The Light is off

Để khai báo các trường hợp của lớp Light, file tiêu đề phải được đưa vào sau đó hai trường hợp đặc biệt được tạo ra. Trường hợp thứ nhất là myLight được xác lập ban đầu sang off do bởi bộ cấu tạo mặc định, thứ hai do Light được xác lập đặc biệt sang on lúc được tạo ra. (hãy nhớ 1 là on và 0 là off).

Các hàm phần tử được viện dẫn thông qua việc sử dụng toán tử dot operator (.) dạng tổng quát là object.function() hoặc object.variable. trong chương trình này ta không thể truy cập trực tiếp trạng thái (giống như trong myLight.status=1) bởi vì biến phần tử được khai báo riêng biệt bằng cách cho phép các biến phần tử chỉ được thay đổi bằng các hàm phần tử thì lớp này tự bảo vệ chính mình khỏi những thay đổi tình cờ hoặc khỏi nhập dữ liệu sai.

Nếu chương trình gọi muốn khai báo một mảng 10 bóng đèn, mỗi hạng mục trong mảng phải được truy cập thông qua các subscript. Phần khai báo sẽ giống như ở dưới đây:

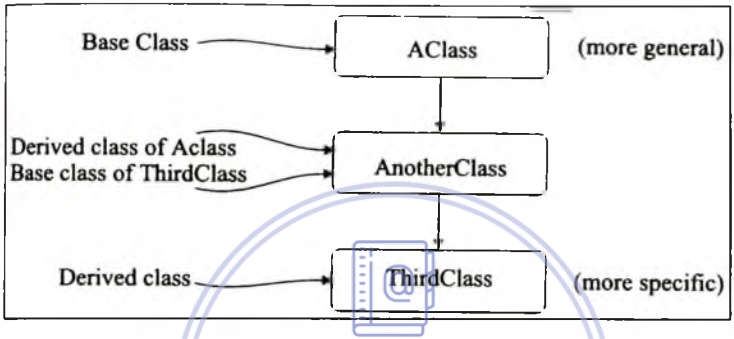
```
Light myLights[10];
```

Sau khi xác lập một vài giá trị, mỗi một bóng đèn riêng biệt có thể được truy cập và được in bằng cách sử dụng vòng lập.

```
for (j=0; j<10; j++) myLights[j].printStatus();
```

8.5.4 Tính kế thừa trong C++

C++ cho phép tính kế thừa, ở đó các lớp con có thể rút ra từ một siêu lớp như minh họa trong hình 8-9. các lớp được rút ra sẽ kế thừa các biến phần tử và các hàm từ lớp bố.

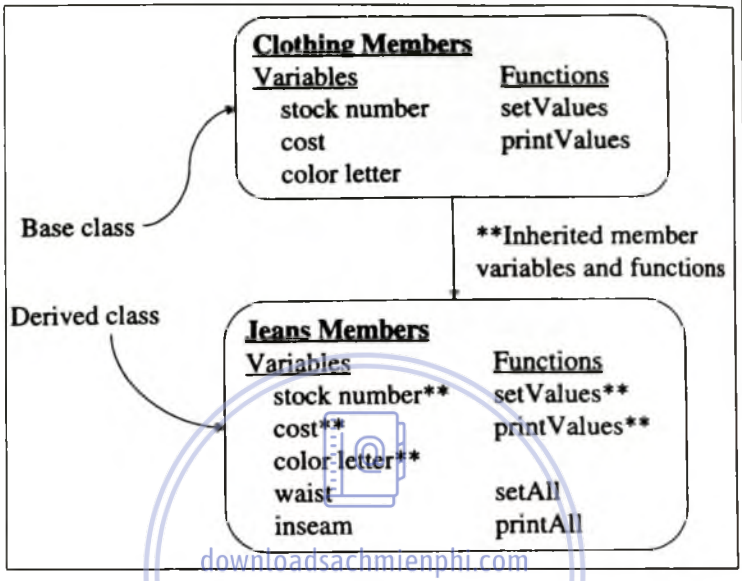


Hình 8-9. *Cơ sở và các lớp được rút ra*

Tuy nhiên, nếu bất cứ phần tử nào được khai báo dưới dạng private thì thậm chí lớp suy dẫn không thể truy cập chúng. Ví dụ, một chiếc xe có thể có một biến phần tử là màu thừa kế từ lớp bố là xe hơi mô tô nhưng nó không thể truy cập giá trị đó với các hàm phần tử được. Nếu các phần tử của lớp bố được khai báo dưới dạng công cộng thì lớp suy dẫn có thể truy cập chúng, nhưng bất cứ đoạn mã nào khác cũng có thể làm việc đó được, và phần tử bảo vệ và dữ liệu ẩn sẽ bị mất. Do đó C++ cho phép một mức khác chuyên biệt cho truy cập bên cạnh mức public và private để được bảo vệ. Lúc phần tử được khai báo dưới dạng protected thì nó có thể đượcthừa kế và được truy cập bất kỳ lớp con nào, nhưng nó vẫn không thể được truy cập từ các đoạn mã khác.

VÍ DỤ 8.7 *Cấu tạo một ADT Jeans là một kiểu chuyên biệt hơn Clothing.*

Để minh họa khái niệm kế thừa trong C++ hãy khảo sát hình 8-10. Clothing sẽ là lớp bố và Jeans sẽ là một lớp suy dẫn. Các biến phần tử có thể có và các hàm được minh họa như sau đây.



***Hình 8-10** ADT dành cho Clothing và Jeans*

Ở tại một cửa hàng đặc biệt, tất cả quần áo phải có một số để đánh dấu trong kho, giá cả và mã màu. Những biến này sẽ được thừa kế bởi bất cứ lớp con nào được suy dẫn từ lớp Clothing. Chỉ có đồ jeans mới cần phải có một kích cỡ và chiều dài, các áo sơ mi thì cần phải có kích cỡ màu và chiều dài nếp gấp. Các kiểu quần áo khác có thể cần các biến bổ sung.

Để thực thi ADT này trong C++ hãy khảo sát mã trong hình 8-11. Trước tiên Clothing.h và Clothing.cpp được tạo, sau đó là các file Jeans.h và Jeans.cpp được viết. Clothing.h phải được đưa vào trong file Jeans.h để chỉ định nguồn của các phần tử kế thừa rồi được chỉ định trong dòng đầu tiên của định nghĩa lớp dưới dạng sau đây:

```
class subsclassName : classAccess parentClassName
```

Bởi vì các biến trường hợp trong lớp Clothing được bảo vệ nên chúng có thể được truy cập trong lớp Jeans. Hàm Jeans::setAll() có thể có chứa mã:

```
stockNum=n; cost=c; colorID=col;
```

Mã này không nằm ở đây bởi vì cho thấy dễ dàng hơn nhiều để lớp suy dẫn sử dụng hàm setValues(...) của bố. Tuy nhiên, lưu ý rằng

không có các phần tử dữ liệu thừa kế nào được xác lập trong bộ cấu tạo Jeans. Các bộ cấu tạo Clothing tự động được gọi trước tiên lúc đối tượng Jeans khởi động.

```
//Clothing.h-header for superclass for clothing

class Clothing
{
  protected:
        int stockNum;
        double cost;
        char colorID;
  public:
        Clothing();                 //constructor
        void setValues (int numIn, double costIn, char colorIn);
        void printValues();
};      double reportCost();
```

```
//Clothing.cpp-implementation of superclass

#include<iostream.h>
#include"Clothing.h"

Clothing::Clothing()                //constructor-zero values
{       stockNum=0; cost=0.0; colorID='';}

void Clothing::setValues (int numIn, double costIn, char colorIn)
//allow the setting of all Clothing values
{       stockNum=numIn; cost=costIn; colorID=colorIn; }

double Clothing::reportCost()       //report the cost
{ return cost;    }

void Clothing::printValues()        //print out the Clothing Info
{       cout<<"Item "<<stockNum<<" costs "<<cost
        <<" with color code "<<colorID;
}
```

```
//Jeans.h-definition for the Jeans class

#include "Clothing.h"

class Jeans:public Clothing         //indicates superclass
{
  protected:
        int waist;
        int inseam;
  public:
        Jeans();                    //constructor
        void setAll(int n, double c, char col, int w, int i);
        //set all values for Jeans
        void printAll();            //print all info
};
```

Hình 8-11 Lớp Clothing và lớp Jeans

```
//Jeans.cpp-implementation of Jeans class

#include <iostream.h>
#include "Jeans/h"

Jeans::Jeans()                          //constructor
{       waist=0; inseam=0; }

void Jeans::setAll(int n, double c, char col, int w, int i)
//set all values for Jeans
{
  setValues(n, c, col);                 //calls superclass function
  waist=w; inseam=i;                    //sets subclass unique values
}

void Jeans::printALL()                  //print unique info
{       printValues();                  //calls superclass function
        cout<<" waist="<<waist<<" and inseam="<<inseam;
}
```

Hình 8-11 *(tiếp theo)*

Một hàm main() thử nghiệm có thể được viết dưới dạng minh họa trong hình 8-12. Clothing.h thì không được đưa vào trong main bởi vì nó chỉ cần thiết đối với file Jeans.h mà thôi. Lúc đối tượng myPants được khởi động, tl.ì trước tiên bộ cấu tạo Clothing được gọi sau đó là bộ cấu tạo Jeans. Kết quả xuất từ main sẽ là :


```
// main() to test jeans
#include <iostream.h>
#include "Jeans.h"

void main ()
{
    Jeans myPants;
    myPants.setAll(55, 24.99, 'b', 34, 30);     //sets all values
    myPants.printAll();                         //prints all
    cout<<endl;
}
```

Hình 8-12*. Thử nghiệm lớp Jeans*

C++ cho phép các lớp cơ sở và lớp kế thừa. Tuy nhiên, đây không phải là một ngôn ngữ lập trình hướng đối tượng thuần túy bởi vì nó cần đến hàm main để cấp phát các cuộc gọi đến các lớp. Java là một cải tiến chuyên sâu của ngôn ngữ lập trình hướng đối tượng thật sự.

CHỦ ĐIỂM 8.6

CLASSES AND INHERITANCE IN JAVA
Lớp và sự thừa kế trong Java

In C++, programmers can create classes, but the basic unit of each program is the function. In Java, however, everything is a class. Java programmers concentrate on creating classes, which are all based upon the Java superclass Object. These classes perform tasks and send messages to other classes. Java is able to accomplish this because of its unique compiling/interpreting nature.

Before looking at the syntax of Java classes, it is important to point out the two environments under which Java programs run. The source code (both definition and implementation) for each class is saved in a file with the extension Java. The Java file is compiled into files of bytecode with the extension .class. The .class files can then be interpreted on the local machine in two ways. First, Java classes can be run as stand-alone applications. Second, Java classes can be created as applets to run on a Web page (under an HTML document). These two program types are unique classes that inherit methods (behaviors) from different Java objects. The manner of compiling varies depending upon the type of compiler. Interpreting, however, can be accomplished by any Java interpreter on any machine. This is one of the reasons for the portability of Java programs.

8.6.1 Java Applications

First, we will examine a simple Java application class.

EXAMPLE 8.8 Write a Java class which prints out "This is a Java application. It is a class called to perform printing tasks."

```
//First.java - first Java Class to examine

import java.io.*;

class First {
  public static void main ( String args[] ) throws IOException
  {

    System.out.print ("This is a Java application.\n");
    System.out.print ("It is a class called to perform printing tasks.\n");

  }//end main
}//end class
```

Fig. 8-13 First Java application (Firstjava).

Figure 8-13 shows a short introductory Java code section. The Java 1/O libraries are included, so the output can be written directly to the console,

and the "throws 10 Exception" is a standard way of dealing with I/O errors, which is beyond the scope of this book. Just consider this as something that should be typed in the heading of a Java application.

The class contains a main() method, inherited from the Java ***Object*** base class, which executes when the program is run. The heading of the main() allows for the possibility of receiving an array of strings as command-line input from the operating system. The "fin" messages correspond to *endl* in C+ + and send the cursor to the next line. The output of this program would look like this:

```
This is a java application.
It is a class called to perform printing tasks.
```

8.6.2 Java Applets

The second way to program in Java is to create classes which inherit from the Java ***Applet*** class, which, in turn, inherits functionality from the Java ***Object*** base class. These classes are run on Web pages under HTML documents.

EXAMPLE 8.9 Write a Java applet to print the same message as Example 8.8 above.

Figure 8-14 shows the FirstApp. java file written as an applet. Notice that two classes are included: the Applet class and the Graphics class. The paint() function, inherited from the Applet class, uses an object of the Graphics class to print strings to the screen. The numbers following the strings give the x and y locations on the screen.

```
//FirstApp.java - first Java Applet to examine

import java.applet.Applet;   //import Applet class
import java.awt.Graphics;    //import Graphics class for writing

public class FirstApp extends Applet {
  public void paint( Graphics g)
  {

     g.drawString("This is a Java applet.", 25, 25);
     g.drawString
          ("It is a class called to print on a Web page.", 25, 40);

  }//end paint
}//end class
```

Fig. 8-14 First Java applet (FirstApp java).

Once the Java applet has been compiled into a class, it can be viewed on any Web browser by a call within an HTML file, as shown in Fig. 8-15. The width and height of the applet space must be enough to contain everything the applet will print to the screen.

```
<html>
<applet code="FirstApp.class" width=275 height=70>
</applet>
</html>
```

Fig. 8-15 HTML file to call applet (First.html).

The output of this short HTML file would look like this:

8.6.3 Using Classes in Java

Besides the application and applet classes, other classes can be created as in C++ and then used by the applications and applets.

EXAMPLE 8.10 Write the Light class from Example 8.5 in Java.

The *Light.* Java file is shown in Fig. 8-16. Java allows the use of boolean variables. Most classes used by Java applets should have a way of returning the instance variables as strings, as in **toStringQ,** because the Graphics obj ect prints only strings to the screen. Also, notice that, as in C++, the variables are private and the methods are public.

```
//class definition (Light.java file)
public class Light
{
    //instance variables
    private boolean status;              //Light is on(true) or off(false)

    //methods
    public Light()                       //default constructor
    { status=false; }                    //false means off, begin with it off

    public Light(boolean statusIn)       //constructor to specific status
    { status=statusIn; }
```

```
    public Light(boolean statusIn)      //constructor to specific status
    { status=statusIn; }

    public boolean flipSwitch()         //change the status
    {   status=!status;                 //if it is off, turn it on. if it is on, turn it off
        return status;
    }

    public String toString()            //returns the current status
    {
        if (status) return "on";
        else return "off";
    }
}
```

Fig. 8-16 Light.java.

An applet that could use the *Light.java* class is shown in Fig. 8-17. First, objects of class Light are declared. In the *init()* method inherited from the Applet class, the objects are instantiated through the use of new. Then the objects can be used in the *paint()* method.

```
//LightApp.java - applet to use Light class

import java.applet.Applet;
import java.awt.Graphics;

public class LightApp extends Applet {

    //declare variables
    private Light myLight;
    private Light yourLight;

    public void init()
    {
        myLight=new Light();
        yourLight=new Light(true);
    }//end init

    public void paint( Graphics g )
    {
        g.drawString ("My Light is" +myLight.toString(), 25, 25);
        g.drawString ("Your Light is" +yourLight.toString(), 25, 40);

        myLight.flipSwitch();
        yourLight.flipSwitch();

        g.drawString ("My Light is" +myLight.toString(), 25, 55);
        g.drawString ("Your Light is" +yourLight.toString(), 25, 70);

    }//end paint
}
```

Fig. 8-17 LightApp lava.

Remember, the applet must run within an HTML document. The Light.html is shown in Fig. 8-18, along with the output that would be shown in the Web browser.

```
<html>
<applet code='LightApp.class' width=275 height=100>
</applet>
</html>
```

```
My Light is off
Your Light is on
My Light is on
Your Light is off
```

Fig. 8-18 Light.html and output.

This chapter has presented an introduction to object-oriented programming and its advantages. Visual Basic operates in an object-oriented environment, C++ allows a hybrid between **OOP and** procedural programming, and Java extensively uses classes. There are some languages like Smalltalk, developed by Xerox at Palo Alto Research Center (PARC), which are pure object-oriented languages. Since each language implements OOP concepts differently, more extensive explanations should be incorporated into the study of a particular language.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 8.6

8.6 CÁC LỚP VÀ TÍNH THỪA KẾ TRONG JAVA

Trong C++ các nhà lập trình có thể tạo các lớp nhưng đơn vị căn bản của một chương trình đó là hàm. Tuy nhiên, trong Java mọi thứ đều là một lớp. Các nhà lập trình Java tập trung về việc tạo ra các lớp, chúng đặt cơ sở trên siêu lớp Java là Object. Những siêu lớp này thực hiện các tác vụ và gửi thông điệp đến các lớp khác. Java có thể hoàn tất điều này do bởi bản chất diễn dịch và biên soạn đơn thuần của nó.

Trước khi xem cú pháp của các lớp Java, điều quan trọng đó là chúng ta phải chỉ ra hai môi trường qua đó các chương trình Java chạy được mã nguồn. (cả định nghĩa lẫn thực thi) dành cho mỗi một lớp phải được lưu trong một file có thành phần mở rộng là .java. file .java được biên soạn thành các file hoặc các mã byte với phần mở rộng là .class. các file .class sau đó có thể được diễn dịch trên máy cục trên hai cách. Trước tiên các lớp Java có thể chạy dưới dạng các trình ứng dụng đứng riêng. Thứ hai các lớp Java có thể được tạo ra dưới dạng các

trình ứng dụng nhỏ để chạy trên một trang web (dưới một tài liệu HTML) hai kiểu chương trình này là các lớp thuần nhất vốn kế thừa các phương pháp (các đặc tính) từ các đối tượng Java khác. Tính cách của việc biên soạn sẽ thay đổi phụ thuộc vào kiểu trình biên soạn. Tuy nhiên, sự diễn dịch có thể được hoàn thành bởi bất cứ trình diễn địch Java nào nằm trong bất cứ máy nào. đây là một trong những lý do khiến cho các chương trình Java mang tính sinh động.

8.6.1 Các trình ứng dụng Java

Trước tiên chúng ta phải xem xét một lớp ứng dụng Java đơn giản.

***VÍ DỤ 8.8** Hãy viết một lớp Java để in ra "This is a Java application. It is a lass called to perform printing tasks". (đây là một trình ứng dụng Java. Nó là một lớp được gọi để thực hiện các tác vụ in).*

Hình 8-13 minh họa một mục mã Java giới thiệu ngắn gọn. Các thư viện I/O của Java được đưa vào vì vậy kết quả xuất phải được viết trực tiếp vào console và "throws IO Exception" là một cách chuẩn để xử lý các lỗi I/O nó vượt xa phạm vi khảo sát của sách này. Ở đây chúng ta chỉ khoan xét một số vấn đề được gõ nhập trong tiêu đề của trình ứng dụng Java.

```
//First.java - first Java Class to examine

import java.io.*;

class First {
  public static void main ( String args[] ) throws IOException
  {
    System.out.print ("This is a Java application.\n");
    System.out.print ("It is a class called to perform printing tasks.\n");

  }//end main
}//end class
```


***Hình 8-13.** Trình ứng dụng Java dầu tiên (First.java)*

Lớp này có chứa một phương pháp main() được kế thừa từ lớp cơ sở Java Object, nó sẽ thực thi lúc chương trình chạy tiêu đề của main() cho phép có thể nhận một mảng của các chuỗi dưới dạng dòng lệnh nhập từ một hệ điều hành. Thông điệp "\n" tương ứng với endl trong C++ và gửi cursor đến dòng kế tiếp. Kết quả xuất của chương trình này sẽ giống như dưới đây.

This is a java application
It is a class called to perform printing tasks.

8.6.2 Các trình ứng dụng nhỏ của Java

Cách thứ hai để lập trình trong Java đó là tạo lớp vốn kế thừa từ lớp Java Applet, qua đó lần lượt kế thừa chức năng từ lớp cơ sở Java Object. Những lớp này được chạy trẽb các trang web dưới các tài liệu HTML.

***VÍ DỤ 8.9** Hãy viết một trình ứng dụng Java để in các thông điệp giống hệt như trong ví dụ 8.8 trên đây.*

Hình 8-14 minh họa file FristApp.java được viết dưới dạng một trình ứng dụng nhỏ. Lưu ý rằng cả hai lớp được đưa vào đó là lớp Applet và lớp Graphic. Hàm paints() được kế thừa từ lớp Applet, sử dụng một đối tượng của lớp Graphic để in các chuỗi trên màn hình. Những con số theo sau các chuỗi cho ta các vị trí x và y trên màn hình.

```
//FirstApp.java - first Java Applet to examine

import java.applet.Applet;   //import Applet class
import java.awt.Graphics;    //import Graphics class for writing

public class FirstApp extends Applet {
  public void paint( Graphics g)
  {

     g.drawString("This is a Java applet.", 25, 25);
     g.drawString
          ("It is a class called to print on a Web page.", 25, 40);
  }//end paint
}//end class
```

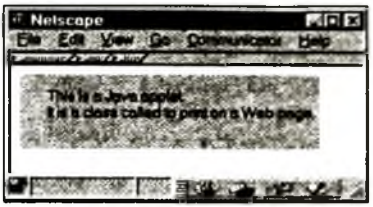

***Hình 8-14.** Trình ứng dụng Frist Java (FirstApp.java)*

Một khi trình ứng dụng Java được biên soạn vào một lớp nó có thể được xem trên bất cứ trình duyệt web nào bởi một cuộc gọi bên trong file HTML như minh họa trong hình 8-15. Chiều rộng và chiều cao của không gian applet phải đủ để chứa mọi thứ mà trình ứng dụng sẽ in ra trên màn hình.

```
<html>
<applet code="FirstApp.class" width=275 height=70>
</applet>
</html>
```

***Hình 8-15.** File HTML đang gọi applet (First.html)*

Kết quả xuất của file HTML ngắn gọn này sẽ giống như dưới đây:



8.6.3 Sử dụng các lớp trong Java

Bên cạnh các lớp trình ứng dụng và ứng dụng nhỏ, các lớp khác có thể được tạo ra như trong C++ và sau đó được dùng bởi các trình ứng dụng và applet này.

***VÍ DỤ 8.10** Hãy viết lớp Light từ ví dụ 8.5 trong Java*

File Light.java được minh họa trong hình 8-16. Java cho phép sử dụng các biến của toán tử boole. Hầu hết các lớp được các trình ứng dụng nhỏ Java sử dụng đều có một cách để trả về các biến trường hợp dưới dạng các chuỗi như trong toString() bởi vì đối tượng Graphic chỉ in các chuỗi sang màn hình. Cũng cần lưu ý rằng như trong C++, các biến là riêng tư và các phương pháp là công cộng.


```
//class definition (Light.java file)
public class Light
{
    //instance variables
    private boolean status;              //Light is on(true) or off(false)

    //methods
    public Light()                       //default constructor
    { status=false; }                    //false means off, begin with it off

    public Light(boolean statusIn)       //constructor to specific status
    { status=statusIn; }

    public boolean flipSwitch()          //change the status
    {   status=!status;                  //if it is off, turn it on. if it is on, turn it off
        return status;
    }

    public String toString()             //returns the current status
    {
        if (status) return "on";
        else return "off";
    }
}
```

***Hình 8-16** Light.java*

Một applet có thể sử dụng lớp Light.java được minh họa trong hình 8.17. Đầu tiên, các đối tượng của lớp Light được khai báo. Trong phương pháp Init() thừa kế từ lớp Applet, các đối tượng được giải

thích qua việc sử dụng new. Sau đó các đối tượng có thể được sử dụng trong phương pháp paint().

```
//LightApp.java - applet to use Light class

import java.applet.Applet;
import java.awt.Graphics;

public class LightApp extends Applet {

   //declare variables
   private Light myLight;
   private Light yourLight;

   public void init()
   {
      myLight=new Light();
      yourLight=new Light(true);
   }//end init

   public void paint( Graphics g )
   {
      g.drawString ("My Light is" +myLight.toString(), 25, 25);
      g.drawString ("Your Light is" +yourLight.toString(), 25, 40);

      myLight.flipSwitch();
      yourLight.flipSwitch();

      g.drawString ("My Light is" +myLight.toString(), 25, 55);
      g.drawString ("Your Light is" +yourLight.toString(), 25, 70);

   }//end paint
}
```


***Hình 8.17** LightApp.java*

Hãy nhớ rằng trình applet này phải chạy bên trong một tài liệu HTML. Light.html được minh họa trong hình 8-18 cùng với kết quả xuất minh họa trong bộ trình duyệt web.

```
<html>
<applet code="LightApp.class" width=275 height=100>
</applet>
</html>

My Light is off
Your Light is on
My Light is on
Your Light is off
```

***Hình 8-18** Light.html và kết quả xuất*

Chương này đã trình bày phần giới thiệu về ngôn ngữ lập trình hướng đối tượng và các ưu điểm của nó. Như Visual Basic hoạt động trong môi trường hướng đối tượng, C++ cho phép một sự lai tạo giữa OOP và chương trình thủ tục, còn Java mở rộng cách sử dụng các lớp cũng có một vài ngôn ngữ chẳng hạn như Smaltalk được phát triển bởi Xerox nằm ở Palo Alto Research Center (PARC) đây thuần túy là các ngôn ngữ hướng đối tượng. Bởi vì mỗi một ngôn ngữ thực thi các khái niệm OOP theo những cách thức khác nhau, phần giải thích chuyên sâu hơn sẽ được liên kết với công trình nghiên cứu ngôn ngữ đặc biệt này.

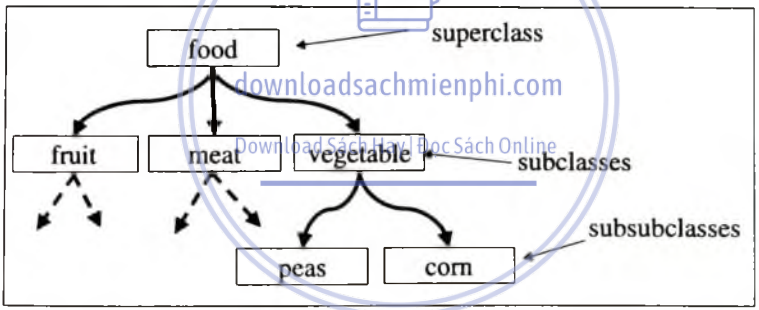
SOLVED PROBLEMS
Bài tập có lời giải

8.1 List possible attributes and behaviors for the following objects: a light switch, a bicycle, an orange, a building, a student.

Some answers, many others are possible:

Object	Possible Attributes	Possible Behaviors
A light switch	status (on, off)	turn on, turn off, check status
A bicycle	model, gear status, color	brake, change gear
An orange	sweetness, color, type	peel, rot, squeeze
A building	temperature, number of floors	turn on heat, open door
A student	name, address, gpa, ss#	change gpa, change address, graduate

8.2 Show the superclass food and some of its subclasses.



8.3 State whether the following applications would normally be examples of procedural or object-oriented programming:

(a) A program asks for the hours worked, calculates the correct amount of pay and prints a check.

(b) A program manipulates a house, allowing the user to set the number of doors and windows, and click a button to draw the model.

(c) A program simulates a college campus, tracking students as the user moves them from building to building.

(d) A program sequentially presents a number of questions and calculates the user's score, displaying the number correct at the end.

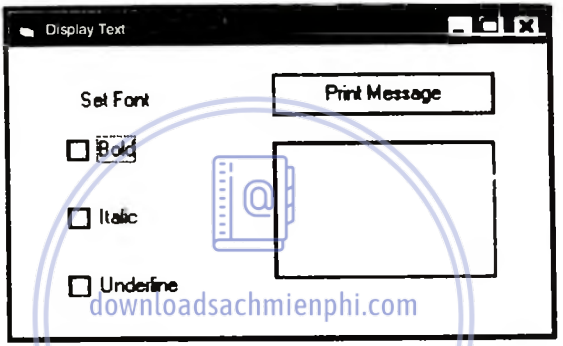
(a) Procedural - the program works sequentially, in the same order every time it is run.

(b) Object-oriented - the attributes of the house object are modified, and then the object displays itself.

(c) Object-oriented - the user instigates the movement and the program responds.

(d) Procedural - the code automatically executes in the same sequence each time it is run.

8.4 Identify the objects on this Visual Basic form.



Answer:

Three checkboxes - Bold, Italic and Underline, used for input choices One label - "Set Font," used to give information One button - "Print Message," used to accept user action One picture box - Empty box often used to print output

8.5 Create the C++ .h file definition for an ADT called Calculator which has two integer member variables called *numl* and *num2,* and one character operator called oper. There should be a default constructor, functions to set the numbers and the operator, a function to perform the calculation, and one to print out the entire problem nicely.

Answer:

```
//Calculator.h
//Calculator class header file

class Calculator
{

   private:
      int numl, num2; //numbers to manipulate
      char oper;      //operation to perform
```

```
    public:
        Calculator();                   //default constructor
        void setNums(int a, int b);     //set the numbers
        void setOper(char operIn);      //set the operation
        int calculate();                //perform operation and return result
        void printCalculation();        //show the current calculation
}; //remember the semicolon
```

8.6 Write the .cpp file for the ADT Calculator. The default constructor should set both numbers to 1 and the operator to '+'. The calculate() function should check for a valid operator and should not try to divide by zero. If there is an error, it can return some flag such as -999. Then the printCalculation() function can call the calculate() function.

Answer.

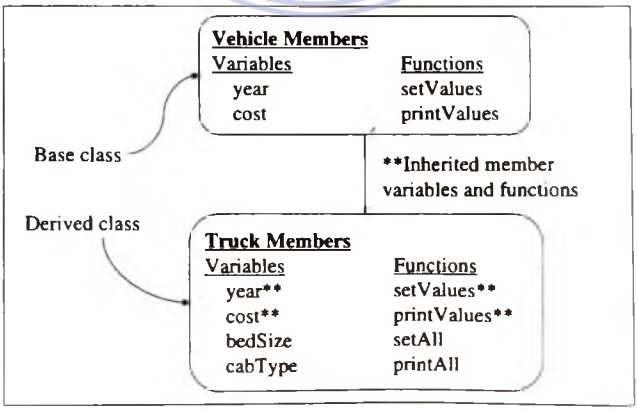
```
//Calculator.cpp
// Calculator class implementation file
#include<iostream.h>
#include "Calculator.h"

Calculator::Calculator()                //default constructor
{     num1=1;
      num2=1;
      oper='+';
}

void Calculator::setNums(int a, int b) //set the numbers to manipulate
{     num1=a; num2=b;}

void Calculator::setOper(char operIn)  //set the operation
{ oper=operIn; }

int Calculator::calculate()             //perform operation and return result
{
    int temp;
    switch (oper)
    {
        case '+': temp=num1+num2; break;
        case '-': temp=num1-num2; break;
        case '*': temp=num1*num2; break;
        case '/': if (num2 !=0) temp=num1 / num2; //integer division
                  else
                  {   temp = -999;
                      cerr<<"Cannot divide by zero"<<endl;
                  }
                  break;
        default: temp = -999;
                 cerr<<"Error in operator"<<endl;
    }// end switch
    return temp;
}

void Calculator::printCalculation() //calculate and show print the entire calculation
{
    char blank=' ';                 //used to help format output
    int temp=calculate();           //calls the member function calculate()
    if (temp>-999)
    cout<<num1<<blank<<oper<<blank<< num2<<blank<<'='<<temp<<endl;
}
```

8.7 Write a short main() program to test the Calculator class. Be sure to try to divide by zero and test an illegal operator.

Sample main(): **Output from each line**

```
#include<iostream.h>
#include "Calculator.h"

void main()
{
    Calculator calc;
    calc.printCalculation();     1 + 1 =2
    calc.setNums(5,3);
    calc.setOper('-');
    calc.printCalculation();     5 - 3 =2
    calc.setOper('*');
    calc.printCalculation();     5 * 3 =15
    calc.setOper('p');
    calc.printCalculation();     Error in operator
    calc.setNums(5,0);
    calc.setOper('/');
    calc.printCalculation();     Cannot divide by zero
    calc.setNums(12,4);
    calc.setOper('/');
    calc.printCalculation();     12 / 4 =3
}
```


8.8 Create the .h file for C++ base class Vehicle, and a derived class Truck from the following specifications. The truck may have a bedsize of 6 or 8 feet, and a cabType of supercab (s) or regular (r).



Code:

```
//Vehicle.h - header for superclass for Vehicle

class Vehicle
{
   protected:
      int year;
      double cost;
   public:
      Vehicle();                  //constructor
      void setValues (int yearIn, double costIn);
      void printValues();
};
//Truck.h - definition for the Truck class inheriting from Vehicle

#include "Vehicle.h"

class Truck : public Vehicle
{
   protected:
      int bedSize;
      char cabType;

   public:
      Truck();                    //constructor
      void setAll(int n, double cstIn, int b, char cab);
                                  //set all values for Truck
      void printAll();            //print all info
};
```


8.9 Create the .cpp files for both classes in Solved Problem 8.8.

```
//Vehicle.cpp - implementation of superclass

#include<iostream.h>

#include "Vehicle.h"
Vehicle::Vehicle()                //constructor - zero values
{     year=0; cost=0.0; }

void Vehicle::setValues (int yearIn, double costIn)
                                  //allow the setting of all Vehicle values
{     year=yearIn; cost=costIn; }

void Vehicle::printValues()       //print out the Vehicle Info
{     cout<<"Vehicle of year "<<year<<" costs "<<cost;
}

// Truck.cpp - implementation of Truck class

#include<iostream.h>
#include "Truck.h"
```

```
Truck::Truck()                    //constructor
{     bedSize=0; cabType=' '; }

void Truck::setAll(int n, double cstIn, int b, char cab)
// set all values for Truck
{
   setValues(n, cstIn);           //calls superclass function
   bedSize=b; cabType=cab;        //sets subclass unique values
}

void Truck::printAll()            //print unique info
{
   printValues();                 //calls public function from superclass
   cout<<"Bed size="<<bedSize<<" and cab type is "<<cabType;
}
```

NOTE: The function setAll() should contain the code to check for proper data values.

8.10 Write a short Java application class to read in a person's favorite digit and print out a message.

```
//Message.java Simple Java application class

import java.io.*;

class Message{
  public static void main ( String args[] ) throws IOException
  {
    //declare variables
    int digit;
    //1. Get the number
    System.out.print ("Enter your favorite digit=>");
    System.out.flush();
    digit=System.in.read();
    digit-='0';

    //3. Print out the message
    System.out.println ("I'm glad you like the number "+digit);
  }//end main
}// end class
```


8.11 Create a Java class to implement the Furnace in Fig. 8-2, and a Java applet to implement the Thermostat that sends messages to the Furnace.

(a) The Furnace object has variables to show whether the heat and air conditioning are on or off. The constructor starts the object with either turned on. There are methods to set the heat on or off, to set the air conditioning on or oft, and to report the string values of the head and air conditioning.

```
// class definition (Furnace.java file)
public class Furnace
{
    //instance variables
    private boolean heat;          //Furnace is on(true) or off(false)
    private boolean ac;            //Air Conditioning is on or off
    // methods
    public Furnace()               //default constructor
    {
        heat=false;                //false means off,
        ac=false;                  //begin with both off
    }
    public void setHeat(boolean status)  //change the status
    {   heat=status;                     //set heat
        ac=false;                        //if heat is turned on or off, turn off ac
    }
    public void setAC(boolean status)    //change the status
    {   ac=status;                       //set AC
        heat=false;                      //if AC is turned on or off, turn off ac
    }
    public String toString()       //returns the current status
    {
       String message;
       if (heat) message="heat on";
       else message="heat off";
       if (ac) message+=" and AC on";
       else message+=" and AC off";
       return message;
    }
}
```


(b) The Thermostat applet object sets all the necessary constants including the desired setting and declares variables representing the current temperature and the Furnace setting. The *init()* sets the initial values and instantiates the myFurnace object. The *paint()* allows for 15 time periods and calls the *CheckAndChangeTemperature()* function, which checks the current temperature and adjusts the heat and air conditioning by sending messages to the Furnace object.

```
//ThermoApp.java - applet to use Furnace class

import java.applet.Applet;
import java.awt.Graphics;

public class ThermoApp extends Applet {

    //declare constants
    final int NOTHINGON=0;
    final int HEATON=1;
    final int ACON=-1;
```

```
    final int DESIREDTEMP=72;
    final boolean ON=true;
    final boolean OFF=false;

    //declare variables
    private Furnace myFurnace;
    int currentTemp;
    int setting;

public void init()
{
   myFurnace=new Furnace();
   currentTemp=64;
   setting=NOTHINGON;
}// end init

public void CheckAndChangeTemperature() //check and set Furnace
{// temperature changes according to what is currently turned on
   if (setting==ACON) currentTemp--;
   if (setting==HEATON) currentTemp++;

   if (currentTemp<DESIREDTEMP)
   {
     myFurnace.setHeat(ON);
     setting=HEATON;
   }
   if (currentTemp>DESIREDTEMP)
   {
     myFurnace.setAC(ON);
     setting=ACON;
   }
}

public void paint( Graphics g )
{
   for (int i=1; i<16; i++)
   {
     g.drawString ("At time "+i+" the temperature is "
          +currentTemp+" with " +myFurnace.toString(), 25, i*15);
     CheckAndChangeTemperature();
   }
}//end paint
}//end class
```

(c) The output of this applet embedded in an HTML document would be:

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

8.1 Liệt kê các thuộc tính và đặc điểm có thể có của các đối tượng sau đây: một công tắc bóng đèn, một xe đạp, một quả cam. Một công trình xây dựng, một sinh viên.

8.2 Biểu thị siêu lớp food và một vài lớp con của nó.

8.3 Phát biểu cho biết các chương trình ứng dụng sau đây có phải là những ví dụ của lập trình hướng đối tượng hay là lập trình thủ tục.

(a) Một chương trình để hỏi giờ làm việc, tính lượng tiền chi trả và in một ngân phiếu.

(b) Một chương trình xử lý một căn nhà cho phép người dùng lắp đặt số cửa và cửa sổ. Nhấp lên một nút để vẽ một mô hình.

(c) Một chương trình mô phỏng một trường học theo dõi học viên khi người dùng di chuyển từ nhà này đến nhà khác. Một chương trình trình bày tuần tự một số các câu hỏi và tính điểm số của người dùng, hiển thị con số chính xác ở cuối.

8.4 Nhận biết các đối tượng trong dạng Visual Basic

8.5 Tạo định nghĩa file C++ .h dành cho một ADT được gọi là Calculator nó có hai biến phần tử nguyên có tên là num1 và num2 và

một toán tử ký tự có tên là oper. Sẽ có ít nhất một bộ cấu tạo mặc định, các hàm để cài đặt số và toán tử, một hàm để thực hiện phép tính và một hàm để in toàn bộ bài toán.

8.6 Hãy viết file .cpp dành cho ADT Calculator. Bộ cấu tạo mặc định nên cài đặt những con số sang 1 và toán tử sang một phép cộng (+). Hàm calculator nên kiểm tra toán tử đúng và không thử chia cho 0. nếu có lỗi nó trả về một số cờ chẳng hạn như –999. Sau đó hàm printcalculator() có thể gọi hàm calculate().

8.7 Hãy viết một chương trình main() ngắn gọn để thử nghiệm lớp Calculator bảo đảm phải thử chia cho 0 và thử nghiệm một toán tử không hợp lý.

8.8 Hãy tạo file .h dành cho lớp cơ sở C++ Vehicle và một lớp suy dẫn Truck từ các đặc trưng sau đây. Truck có thể có kích cỡ từ 6 hoặc 8 feet và cabType của supercab(s) hoặc regular (r).

8.9 Tạo các file .cpp dành cho cả hai lớp trong bài tập có lời giải 8.8

8.10 Hãy viết một lớp trình ứng dụng Java ngắn gọn để đọc trong một chữ số của một người rồi in ra thông điệp.

8.11 Tạo một lớp Java để thực thi Furnace trong hình 8-2 và một trình ứng dụng nhỏ Java để thực thi Thermostat để qua đó gửi thông điệp đến Furnace.

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

8.12 List possible attributes and behaviors for each object in the chart below.

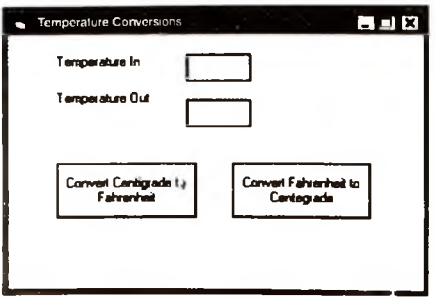
Object	Possible Attributes	Possible Behaviors
A circle A rectangle A book A truck An employee		

8.13 Show a superclass *reading material* and some of its subclasses.

8.14 State whether the following would normally be examples of procedural or object-oriented programming:

Application	Programming Type
(*a*) A program monitors the number of cars passing an intersection, adding one every time a car runs over an input cord.	
(*b*) A program maintains inventory week! ; by getting input from a transaction file, and adding or subtracting items from a master inventory list.	
(*c*) A program allows users to play solitaire, responding to clicks on the cards on the screen.	
(*d*) A program mimics a vending machine, counting the money entered, and dispensing the drink requested.	

8.15 Identify the objects on this Visual Basic form.

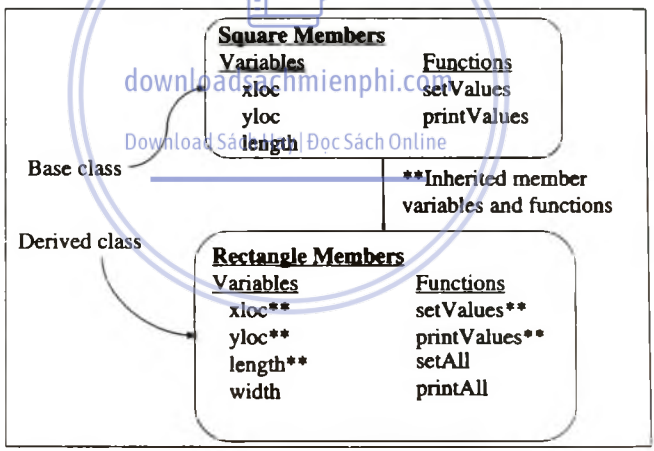


8.16 Create the .h file definition for an ADT called *Student*. Member variables should include an integer ID, an integer array of 10 test scores, and an integer which records how many tests are entered into the array. Member functions should include a default constructor, functions to set the ID, to add scores to the list, to calculate and return a floating point number for the average score, and to print out all the student information.

8.17 Create the .cpp file for all the functions in the ADT *Student*. The default constructor should set the ID, the number of scores, and all the scores in the array to zero. The function to calculate the average should return zero if there are no scores in the array.

8.18 Create a main() function to test the *Student* class which creates an array of five students, sets their IDs to the integers 0 through 4, adds some dummy scores and prints the information for the students in the array.

8.19 Write the .h files to implement the base class Square and the derived class Rectangle as defined below. The Square has the x and y locations of the upper left corner, and the length of each side. The Rectangle inherits those values and adds a width.



8.20 Write the .cpp files for the Square and Rectangle classes.

8.21 Write a simple Java application class that will read in one digit and print its powers from 0 to 10. Show the output of your program with the digit 2 entered.

8.22 Create a Java class Car, which has variables of *model*, *year*, and *mpg*. Write a Java applet Trip that will print out the distance two cars could go if the tank had 2, 4, 6, or 8 gallons in it. Show the output for cars with 25 and 35 miles per gallon.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

8.12 Hãy liệt kê các thuộc tính và các đặc điểm có thể có ứng với mỗi một đối tượng trong sơ đồ dưới đây.

8.13 Biểu thị một siêu lớp reading material và một số lớp con của nó.

8.14 Phát biểu cho biết các ví dụ sau đây là lập trình thủ tục hay là lập trình hướng đối tượng.

8.15 Nhận biết các đối tượng trong dạng Visual Basic.

8.16 Tạo định nghĩa file .h dành cho ADT có tên là student. Các biến phần tử nên có chứa một ID nguyên, một mảng nguyên 10 điểm thử nghiệmvà một số nguyên để ghi số lần thử nghiệm được nhập vào trong mảng. Các hàm phần tử nên có chứa một bộ cấu tạo mặc định, các hàm số để xác lập ID, để bổ sung điểm số vào danh sách, để tính và trả về một số chấm động dành cho điểm trung bình và để in tất cả các thông tin của học viên

8.17 Tạo file .cpp dành cho tất cả các hàm trong ADT student. Bộ cấu tạo mặc định nên xác lập ID, các điểm số và tất cả các điểm trong mảng sang 0. Hàm tính điểm trung bình nên trả về 0 nếu không có điểm nào trong mảng.

8.18 Tạo một hàm main() để thử nghiệm lớp student qua đó tạo một mảng 5 học viên, xác lập ID của chúng sang các số nguyên từ 0 – 4, bổ sung một vài điểm và in thông tin dành cho học viên trong mảng.

8.19 Hãy viết các file .h để thực thi lớp cơ sở Square và lớp suy dẫn Rectangle như được định nghĩa dưới đây. Square có các vị trí x và y nằm ở góc bên trái phía trên và chiều dài của mỗi cạnh. Rectangle kế thừa các giá trị và thêm vào một chiều rộng.

8.20 Hãy viết các file .cpp dành cho các lớp Square và Rectangle.

8.21 Hãy viết một lớp ứng dụng Java đơn giản sẽ đọc một chữ số và in số mũ của nó từ 0 – 10. Hãy biểu thị kết quả xuất chương trình của bạn với hai chữ số 2 được nhập vào.

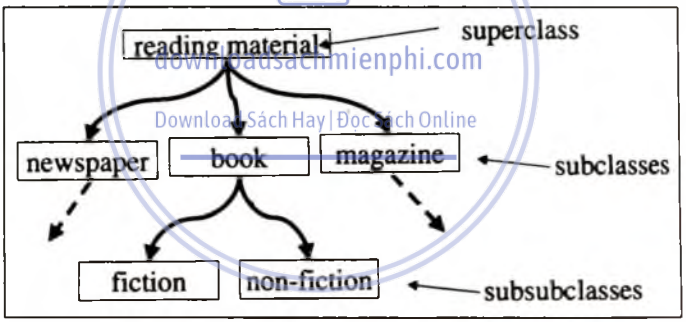
8.22 Tạo một lớp Java Car có các biến model, year và mpg. Hãy viết một trình ứng dụng Java Trip để in ra khoảng cách giữa hai xe nếu thùng có chứa 2, 4, 6, 8 gallon trong đó. Biểu thị kết quả xuất dành cho các xe hơi chạy 25 và 35 dặm trên mỗi gallon.

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

8.12 Some answers, many others would be possible

Object	Possible Attributes	Possible Behaviors
A circle	radius, color, location	calculate area, draw circle, change color, change location
A rectangle	length, width, color, location	calculate area, draw rectangle, change color, change location
A book	number pages, font type and size, words per page	print out, change font and size, change number of pages
A truck	model, color, weight, gear	accelerate, brake, change gears
An employee	name, address, ss#, pay rate, hours worked	change address, change pay rate calculate net pay

8.13 One possible answer



8.14

Application	Programming Type
(*a*) cars	Object-oriented
(*b*) inventory	Procedural
(*c*) solitaire	Object-oriented
(*d*) vending machine	Object-oriented

8.15 Two buttons - "Convert ...."
Two textboxes - empty
Two labels - "Temperature..."

8.16 Student.h:

```
//Student.h - header file for Student class

class Student
{
   private:
      int ID;
      int scores[10];
      int numberScores;
   public:
      Student();                    //default constructor
      void setID(int IDin);         //set the ID
      void addScore(int scoreIn);//add score to list
      float calcAvg();              //calculate and return average score
      void printStuInfo();          //print all current information
};
```

8.17 Student.cpp:

```
//Student.cpp - implementation for class Student

#include<iostream.h>
#include "Student.h"

Student::Student()                  //default constructor
{
   ID=0;
   numberScores=0;
   for (int i=0; i<10; i++) scores[i]=0;
}

void Student::setID(int IDin)       //set the ID
{   ID=IDin;    }

void Student::addScore(int scoreIn) //add score to list
{
   scores[numberScores]=scoreIn;
   numberScores++;                  //add 1 to number of scores
}

float Student::calcAvg()            //calculate and return average score
{
   float avg=0, sum=0;
   if (numberScores>0)
   {
      for (int i=0; i<numberScores; i++) sum +=scores[i];
      avg=sum / numberScores;
   }
   return avg;
}
```

```
void Student::printStuInfo()          //print all current information
{
    cout<<'Student number '<<ID<<' had '<<numberScores<<
        ' grades and an average of '<<calcAvg()<<endl;
}
```

8.18 Main() to test Student class:

```
//main() to test Student class

#include<iostream.h>
#include "Student.h"

void main()
{
    int i, j;
    Student myClass[5];                //declare an array of students
    for (i=0; i<5; i++)
    {
        myClass[i].setID(i);           //set ID equal to i
        for (j=0; j<i+1; j++)
            myClass[i].addScore(j+3); //add some dummy scores
        myClass[i].printStuInfo();     //print the info
    }
}
```


Output from the main() above:

Student number 0 had 1 grades and an average of 3
Student number 1 had 2 grades and an average of 3.5
Student number 2 had 3 grades and an average of 4
Student number 3 had 4 grades and an average of 4.5
Student number 4 had 5 grades and an average of 5

8.19 Header files for Square and Rectangle:

```
//Square.h - header for superclass for Square

class Square
{
   protected:
      int xloc;
      int yloc;
      double length;

   public:
      Square();                        //constructor
      void setValues (int xIn, int yIn, double lengthIn);
      void printValues();
```

```
//Rectangle.h - definition for the Rectangle class inheriting from Square
#include "Square.h"

class Rectangle : public Square
{
   protected:
      double width;

   public:

      Rectangle();                        //constructor
      void setAll(int x,int y, double l, double w);
                                          //set all values for Rectangle
      void printAll();                    //print all info
};
```

8.20 Implementation files for Square and Rectangle:

```
//Square.cpp - implementation of superclass

#include<iostream.h>
#include "Square.h"

Square::Square()                          //constructor - zero values
{   xloc=0; yloc=0; length=0.0; }

void Square::setValues (int xIn, int yIn, double lengthIn)
// allow the setting of all Square values
{   xloc=xIn; yloc=yIn; length=lengthIn; }

void Square::printValues() // print out the Square Info
{   cout<<"Four sided shape at location x= "<<xloc<<" y= "<<yloc
    <<" with side lengths of "<<length;
}
//Rectangle.cpp - implementation of Rectangle subclass

#include<iostream.h>
#include "Rectangle.h"

Rectangle::Rectangle()                    //constructor
{   width=0.0; }

void Rectangle::setAll(int x,int y, double l, double w)
//set all values for Rectangle
{
   setValues(x, y, l);                    //calls superclass function
   width=w;                               //sets subclass unique values
}

void Rectangle::printAll()                //print unique info
{
   printValues();                         //calls public function from superclass
   cout<<" and width of "<<width;
}
```

8.21 PowerIt class:

```
//PowerIt.java - class gets a digit, calculates and prints its powers from 0 to 10

import java.io.*;

class PowerIt {
  public static void main ( String args[] ) throws IOException
  {
    //declare variables
    int num;
    int product;

    //1. Get the number
    System.out.print ("Enter a number between 0 and 9=>");
    System.out.flush();
    num=System.in.read();
    num=num-'0'; // change character to integer
    product=num;

    //2. Print out the 0th power
  System.out.println (num+" to the 0 power is "+1);

    //3. Print out the powers
    for (int i=1; i<=10; i++)
    {
        System.out.println (num+" to the "+i+" power is "+product);
        product *= num;
    }// end for loop
  }// end main
}// end class
```



```
2 to the 0 power is 1
2 to the 1 power is 2
2 to the 2 power is 4
2 to the 3 power is 8
2 to the 4 power is 16
2 to the 5 power is 32
2 to the 6 power is 64
2 to the 7 power is 128
2 to the 8 power is 256
2 to the 9 power is 512
2 to the 10 power is 1024
```

8.22 Implementation for Car and Trip:

```
//class definition (Car.java file)
public class Car
{
```

```
    //instance variables
    private String model;
    private int year;
    private int mpg;

    //methods
    public Car(String m, int y, int mpgIn)   //constructor with values
    {
        model=m;
        year=y;
        mpg=mpgIn;
    }

    public int getMPG()                      //sends back private variable mpg
    { return mpg; }

    public String toString()                 //returns the current status
    {
        String message;
        message="The car is a "+year+" "+model +
            " which gets "+mpg+" miles per gallon";
        return message;
    }

}
//TripApp.java - applet to use Car class

import java.applet.Applet;
import java.awt.Graphics;

public class TripApp extends Applet {

   //declare variables
   private Car myCar, yourCar;

 public void init()
 {
   myCar=new Car("Ford", 1997, 25);
   yourCar=new Car ("Volvo", 1996, 35);
 }// end init
 public void paint( Graphics g )
 { int i, yloc=15;
   g.drawString (myCar.toString(), 25, yloc);
   for (i=2; i<=8; i+=2)
   {
     yloc+=15;
     g.drawString ("With "+i+" gallons it will travel "
          +myCar.getMPG() * i+" miles", 25, yloc);
   }

   yloc+=30;
   g.drawString (yourCar.toString(), 25, yloc);
```

```
        for (i=2; i<=8; i += 2)
        {
            yloc += 15;
            g.drawString ("With "+i+" gallons it will travel "
                    +yourCar.getMPG() * i+" miles", 25, yloc);
        }
    }// end paint
}// end class
```

Output:

CHAPTER 9

Data Structures

Cấu trúc dữ liệu

MỤC ĐÍCH YÊU CẦU

Sau khi học xong chương này, các bạn sẽ nắm vững các vấn đề về cấu trúc dữ liệu, cụ thể với các nội dung cơ bản sau đây:

- Introduction to data structures
- Linked lists
- Stacks
- Queues

- *Giới thiệu về cấu trúc dữ liệu*
- *Danh sách liên kết*
- *Xếp chồng (Hàng chồng)*
- *Hàng đợi*

Ngoài ra, ở cuối chương còn có phần bài tập có lời giải, bài tập bổ sung và đáp án nhằm giúp các bạn thực hành và áp dụng một cách hiệu quả vào công việc thực tế.

CHỦ ĐIỂM 9.1

INTRODUCTION TO DATA STRUCTURES
Giới thiệu về cấu trúc dữ liệu

Computer programs are made up of algorithms and data structures. Algorithms have been used extensively in this book to provide the step-by-step instructions of what a program will do. The data or variables are the raw materials with which the algorithms work. A single variable represents one memory location that can hold a variable of a simple type such as integer or character. A **data structure** is a group of memory locations used to represent the information used by the algorithm.

Some data structures have already been examined. For example, an array is a fixed-length data structure that sets aside a group of locations called by one common name, each containing the same type of data. Almost all languages provide for the use of arrays. A class is a data structure called by one name that can contain variables of different types, as well as the group of behaviors possible for that class. Classes are built in to some languages and recognized by the compiler. Sometimes, however, programs require other kinds of data structures that do not exist in a particular language. The programmer can build representations of such structures, and then use them in any program that requires that particular type of data.

This chapter examines three common data structures used by computer scientists: linked lists, stacks, and queues. These structures are not built in to most languages, although they may be included in some supplementary libraries. Usually they must be created. Classes are ideal building blocks for the creation of other structures because the programmer can package all the required variables and behaviors together for use by any program. Therefore, in each case, the structures will be created using classes in C++ or Java.

There are two ways to implement these structures: (1) using static data types (arrays), and (2) using dynamic data types (pointers). Static data types use a fixed amount of memory even if not all locations are filled. Dynamic data types use pointer variables, and grow and shrink during the run of the program through the allocation and deallocation of memory. Linked lists will be described using both static and dynamic structures. However, since dynamic structures are usually employed, stacks and queues will be implemented only dynamically.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 9.1

9.1 GIỚI THIỆU VỀ CẤU TRÚC DỮ LIỆU

Các chương trình máy tính được hình thành từ các thuật toán và các cấu trúc dữ liệu. Các thuật toán được dùng một cách rộng rãi trong sách này để cung cấp các chỉ dẫn theo từng bước mà chương trình sẽ thực hiện. Dữ liệu các biến là dữ liệu thô mà thuật toán phải làm việc. Một biến đơn biểu thị một vị trí bộ nhớ vốn có thể giữ một biến thuộc các kiểu đơn giản chẳng hạn như số nguyên hoặc ký tự. Một cấu trúc dữ liệu là một nhóm dữ liệu trong bộ nhớ dùng để biểu thị thông tin mà thuật toán sử dụng.

Một số cấu trúc dữ liệu đã được kiểm tra. Ví dụ một mảng chính là một cấu trúc dữ liệu có chiều dài cố định được xác lập tách biệt với một nhóm các vị trí và được gọi theo một tên chúng, mỗi vị trí có chứa kiểu dữ liệu giống nhau. Hầu hết các ngôn ngữ đều cung cấp để sử dụng các mã. Một lớp là một cấu trúc dữ liệu được gọi theo một tên và có thể chứa các kiểu biến khác nhau cũng như nhóm các đặc tính có thể có dành cho lớp đó. Các lớp được cấu tạo sẵn cho một vài ngôn ngữ và chỉ được trình biên soạn nhận biết mà thôi. Tuy nhiên, các chương trình yêu cầu các kiểu cấu trúc dữ liệu khác vốn không hiện diện trong một ngôn ngữ đặc biệt. Người lập trình có thể xây dựng các phần trình bày cấu trúc rồi sử dụng chúng trong bất cứ chương trình nào vốn yêu cầu sử dụng dữ liệu đặc biệt đó.

Chương này xem xét ba cấu trúc dữ liệu chung mà các nhà khoa học máy tính sử dụng: các danh sách liên kết, các chồng và hàng đợi. Những cấu trúc này không được cấu tạo sẵn trong hầu hết các ngôn ngữ, mặc dù chúng có thể được đưa vào trong một vài thư viện bổ sung. Thông thường chúng phải được tạo. Các lớp là các khối cấu trúc lý tưởng dành để tạo các cấu trúc khác bởi vì người lập trình có thể đóng gói tất cả các biến và các đặc điểm cần thiết lại với nhau để bất cứ chương trình nào sử dụng thực thi những cấu trúc này được. Do đó, trong mỗi trường hợp, các cấu trúc sẽ được tạo bằng cách sử dụng các lớp trong C++ hoặc trong Java.

Có hai cách để cấu trúc các thực thi này: (1) sử dụng các kiểu dữ liệu tĩnh (mảng), và (2) sử dụng các kiểu dữ liệu động (con trỏ). Các dữ liệu tĩnh sử dụng một lượng bộ nhớ cố định thậm chí khi không phải tất cả các vị trí đều bị làm đầy. Các kiểu dữ liệu động thì sử dụng các biến con trỏ và phát triển cũng như thu gọn lại trong suốt quá trình chạy chương trình thông qua việc cấp phát và hủy cấp phát các bộ nhớ. Các danh sách liên kết sẽ được mô tả bằng cách sử dụng cả cấu trúc tĩnh lẫn cấu trúc động. Tuy nhiên bởi vì các cấu trúc động thường được thực thi do đó các hàng chồng và các hàng đợi được thực thi chỉ ở dưới dạng động mà thôi.

CHỦ ĐIỂM 9.2

LINKED LISTS
Danh sách liên kết

A ***linked*** list abstract data type (ADT) is a list or chain of items where each item points to the next one in the list. Each item in a linked list is called a ***node.*** Each node contains the needed data and also the location of the next item. The data can be in any form, such as an integer, an array, or even another class object, and is usually maintained in some kind of order depending upon the specific application. The content of the link varies depending upon the implementation.

EXAMPLE 9.1 Draw an abstract representation of a single node and a linked list.

Consider the representation shown in Fig. 9-1.

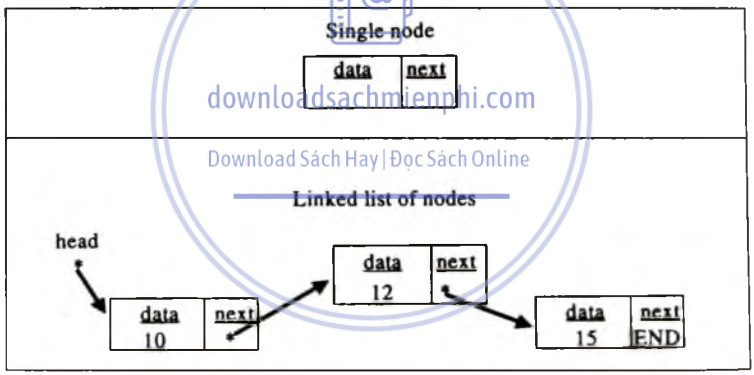


Fig. 9-1 A node and a linked list.

The linked list ADT usually contains member variables for the location of the beginning of the list and the size of the list. The minimum linked list functions needed are to construct the list, insert an item into the list, delete an item from the list, and print the contents of the list. Other functions may be added if needed, such as to report the size of the list, print the kth item, search the list, and so forth. The actual code for insert and delete varies according to whether the list is maintained in order. If the list is in order, an item is inserted in its proper spot. All that is needed is to create the node and then adjust the pointers, as shown in Fig. 9-2.

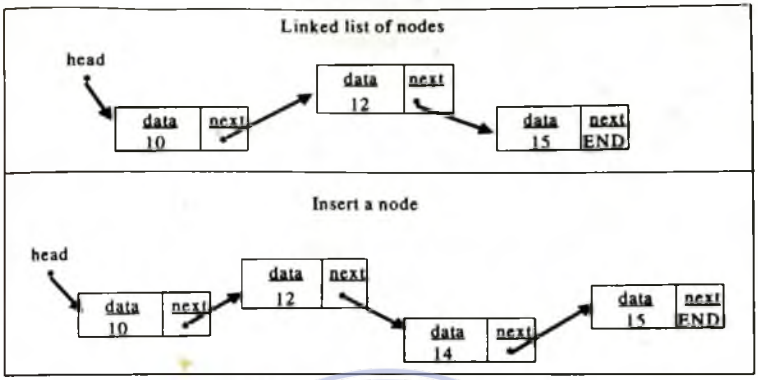


Fig. 9-2 Inserting a node into a linked list.

Linked lists have a distinct advantage over simple arrays. When inserting or deleting an item in an ordered linked list, only the pointers are adjusted. In an array, all items below the insertion point must be moved up or down to keep the proper order. However, in an array, one can find any specific item directly (e.g., arrayName[31]). In order to find kth item in a linked list, the list must be traversed by following the pointer of each node to the location of the next node. It is impossible to find a single item without traversing the entire list. Therefore, arrays are faster for finding a specific item and linked lists are more efficient for insertions and deletions. The specific implementation of most data structures results in some kind of trade-off between space and time.

9.2.1 Static Implementation of Linked List - *Sự thực thi tĩnh của danh sách liên kết*

The technique for implementing a static linked list involves using an array for the list space. The example used in this section is an unordered list, with items inserted at the end of the list. Each item in the array is a node. The link of each node contains the array location of the next item in the list.

EXAMPLE 9.2 Draw a representation of a static linked list. Figure 9-3 shows this example.

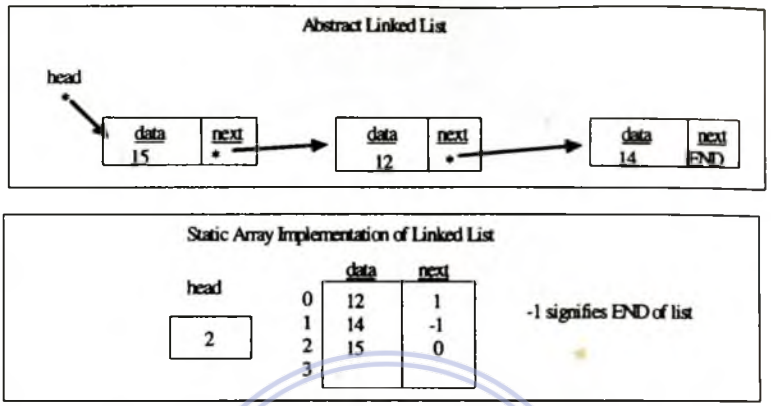


Fig. 9-3 Static linked fist.

Notice two things about this implementation. First, the array space is a pool of available nodes which also is a linked fist. When inserting an item, the first available node is chosen. In Fig. 9-3, if another item is inserted into the list it would be placed in spot 3. The list needs to keep track of the next available node. Second, the array contains a finite number of locations and must be big enough for all possible lists. For example, an array may be used for course fists. The largest course has 100 students and the smallest has 20. Students add to and drop from the list. The given array space might have 200 locations. Therefore, there are three numbers that must be tracked: the maximum available in the array, the maximum to be used in any given list, and the index of the current number in the list.

EXAMPLE 9.3 Implement in C++ a static linked list of integers.

This is implemented in C++ through the use of the LinkedList class. The LinkedList.h file is shown in Fig. 9-4a. All the member variables and functions needed for the list are included. The functions are implemented in Fig. 9-4b and explained below.

```
/*
 *class LinkedList
 */

typedef int Item;
const int MAX=8;              //maximum possible in the array
const int END=-1;             //constant signifies the end of any list

class Node
{
  public:
    Item value;
    int link;
};

class LinkedList
{
  private:
    Node space[MAX];
    int head;      //address of the first node in list
    int avail;     //address of first node available to be used
    int maxSize;   //maximum number of items in the current list
    int size;      //number of items in current list

public:

    LinkedList(int n);          //Initializes the LinkedList as empty with maximum size n.

    int Size() {return size;}; //returns size of list - use of inline function

    void Insert(Item x);        //Inserts an item x in first available spot

    void Delete(Item x);        //Removes the item x - Prints error if x was not in the list

    void PrintList(char mode); //prints the list in two ways for testing - either
                                //mode=(o) list in order or (a) print array contents
};
```

Fig. 9-4a LinkedList.h.

```
//LinkedList.cpp - implementation of LinkedList class

#include<iostream.h>
#include "LinkedList.h"
LinkedList::LinkedList(int n)
//Initializes the LinkedList as empty with maximum size n.
{
    head=END;
    size=0;
    avail=0;
    maxSize=n;
    for (int i=0; i<MAX; i++)       //set up all the spaces
    {     space[i].value=END;
          space[i].link=i+1;
    }

    space[MAX-1].link=END;          //end the total list
    space[n-1].link=END;            //end the list of available nodes for the current list
}
void LinkedList::Insert(Item x)
//insert x in first available spot and hook at end of list
{
```

Fig. 94b LinkedListcpp.

```
    int p, current;
    //get the first spot
  if (avail==END) cout<<"\n LinkedList::Insert -- list is full \n";
  else
  {
    p=avail;
    avail=space[avail].link;        //next available spot
    space[p].value=x;               //fill values
    space[p].link=END;
    //find the end to hook it up
    if (head==END) //hook it at the beginning
        head=p;
    else
    {
        current=head;
        while (space[current].link !=END)
            current=space[current].link;
        //now current points to last item
        //hook it up
        space[current].link=p;
    }//end else hook at end
    size++;
  }//end else list is full
}
  void LinkedList::Delete(Item x)
  {
      //find the item
    int prev, current;
    if (head==END) cout<< "\n LinkedList::Delete - List is empty \n";
    else
    {
  prev=head;
  current=head;
  while (space[current].link !=END && space[current].value !=x)
  {
      prev=current;
      current=space[current].link;
  }
  if (space[current].value==x) //item is found
  {
      //unhook it
      //check to see if it is first in list
      if (current==head) head=space[current].link;
      else space[prev].link=space[current].link;

      //return space to avail
       space[current].link=avail;
       avail=current;
       size--;
  }
  else cout<< "\n LinkedList::Delete "<<x<<" not found \n";
}//end else list is not empty
  }

  void LinkedList::PrintList(char mode)
  //prints the list either mode=(o) list in order or (a) print array contents
  {
      int i=0;
      switch (mode)
      {
```

Fig. 94b (Continued)

```
    case 'o':
        cout<<'\nTHE LIST:\n';
        i=head;
        if (i==END) cout<<'Empty\n';
        else
            while (i !=END)
            {
                cout<<space[i].value<<endl;
                i=space[i].link;
            }
        break;
    case 'a':
        cout<<'\nTHE LIST: starts in spot '<<head<<endl;
        cout<<'The first available spot is '<<avail<<endl;
        for (i=0; i<MAX; i++)
            cout<<'['<<i<<']'<<'\t'<<space[i].value<<'\t'<<space[i].link<<endl;
        break;
    default:
        cout<<'LinkedList: PrintList - error in mode\n';
    }
    cout<<endl;
}
```

Fig. 94b (Continued)

LinkedList(int n): The constructor sets up the needed items for an empty list that may ultimately grow only to size n. The head of the list is set to -1 to signify an empty list. The first available spot is location 0, and the list of available nodes all point to the empty location following. For each node, the value is arbitrarily initialized to END. (The initialization could be set to 0, or 999, or any other number chosen to represent an undefined value.) Finally, the last item in the array and the last item in the available list are both set to END.

Insert(Item x): The Insert function checks to see if the list is full (needed for static structures of finite size), gets the next available spot, adjusts the available list, and then adds the item x at the end of the list. A different algorithm could be written to insert the item at the beginning. Also, if the list were ordered, an algorithm would be written to insert the item into its proper location. These functions are shown in the exercises at the end of this chapter. Since this implementation keeps track of the size of the current list, the size variable is incremented.

Delete(Item x): The Delete function first checks to see if the list is empty. If the list is not empty, the function searches for the item to be deleted. Two external pointers are needed: one to point to the current node and one to point to the previous node. The previous pointer follows the current pointer through the list so that when the item is found it can be defeted through the adjustment of two pointers. See Fig. 9-5 for an example. Once the item is found, the item must be removed from the list and the location returned to the list of available nodes so it can be used again.

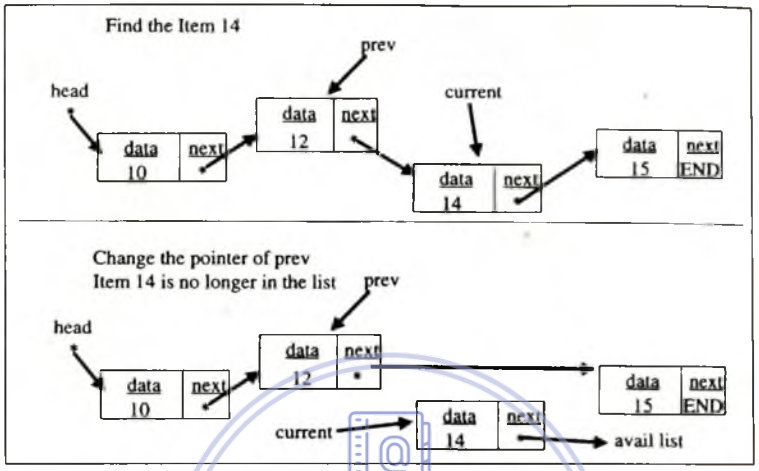


Fig. 9-5 Deleting an item from the list - abstract view.

PrintList(char mode): Usually the Print function would print the contents of the list in order. However, this Print function prints the list in two ways for the purposes of explanation and program testing. When the mode comes in as 'o' the values of the list are printed in the order in which they occur in the list. With the mode as 'a', the entire array is printed. This print function could be tailored to print in whatever way needed by a particular application.

EXAMPLE 9.4 Write a section of C++ code to use the LinkedList structure.

Shown below are some consecutive code sections in a main() function and a picture of the list and the array after each section.

Code	mylist.PrintList('o');	mylist.PrintList('a');

`LinkedList mylist(5);`
 	
//sets up empty list with:
//MAX of array = 8 (see .h file)
//max size of this list = 5
	The list:
	Empty
		The list: starts in spot −1
		The first available spot is 0
		[0] −1 1
		[1] −1 2
		[2] −1 3
		[3] −1 4
		[4] −1 −1
		[5] −1 6
		[6] −1 7
		[7] −1 −1

Code	mylist.PrintList('o');	mylist.PrintList('a');

`mylist.Insert(10);`
`mylist.Insert(14);`
`mylist.Insert(9);`
`mylist.Insert(3);`
`mylist.Insert(15);`
 	
//list is now full, so avail = −1
	The list:
	10
	14
	9
	3
	15
		The list: starts in spot 0
		The first available spot is −1
		[0] 10 1
		[1] 14 2
		[2] 9 3
		[3] 3 4
		[4] 15 −1
		[5] −1 6
		[6] −1 7
		[7] −1 −1

`mylist.Delete(10);`
`mylist.Delete(3);`
 	
/* Start with 1 and follow the links for the list.
Start with 3 and follow the links for the available nodes.
Remember, only 0 through 4 can be used*/
	The list:
	14
	9
	15
		The list: starts in spot 1
		The first available spot is 3
		[0] 10 −1
		[1] 14 2
		[2] 9 4
		[3] 3 0
		[4] 15 −1
		[5] −1 6
		[6] −1 7
		[7] −1 −1

`mylist.Insert(25);` /* Again, start with 1 and follow the links for the list. Start with 3 and follow the links for the available nodes.*/	The list: 14 9 15 25	The list: starts in spot 1 The first available spot is 0 [0] 10 −1 [1] 14 2 [2] 9 4 [3] 25 −1 [4] 15 3 [5] −1 6 [6] −1 7 [7] −1 −1

9.2.2 Dynamic Implementation of Linked List - *Sự thực thi động danh sách liên kết*

The static implementation of the linked list above had two problems. First, the array had to be created large enough for any possible application, even if not all the spaces were used in a given list. Also, the list of available nodes had to be maintained by the class itself. In the dynamic implementation, the available space is all of memory, which is called heap space, and the available locations are tracked by the computer. Nodes are created as needed and returned to the heap space for later use. The list class only needs a pointer to the head of the list and an integer to keep track of the size.

EXAMPLE 9.5 Write the Java code to implement an ordered linked list.

In an ordered list, each node is less than or equal to the node that follows it. In this case, the Node is a separate class that handles its own

operations. These functions allow the LinkedList class to access the Node class variables. There are two constructors: one to create a Node with a null link, and one to create a Node pointing to another node. Figure 9-6a contains the Node class and Fig. 9-6b shows the LinkedList class.

```
//class definition (Node.java file)

public class Node
{
    int value;
    Node link;                      //pointer to itself is implicit in Java

    Node (int x) this(x, null);}  //constructor to create a node with data of x

    Node(int x, Node next) //constructor creates node with data of x and points to next in list
    {
      value=x;
      link=next;
    }

    int getValue() {return value;}              //return the value of the node

    Node getLink() {return link;}               //returns the next node in the list

    void setLink(Node newLink) {link=newLink;}  //resets the value of link
}
```

Fig. 9-6a Java Node dass.

```
//class definition (LinkedList.java file)

public class LinkedList
{
  //instance variables
  private Node head;          //only head of list needed - all memory is available
  private int size;           //current size of list
  //methods
  public LinkedList(){ head=null; }          //default constructor

  int getSize (){ return size; }             //returns size of list

  public void Insert(int x)                  //inserts x and hooks to begining of list
    {    Node p, prev, current;
         if (head==null)                     //list is empty
        head=new Node(x);
        else                                 //insert in order
        {
          //find spot to insert
          current=head;
          prev=head;
          while (current.getLink() !=null && current.getValue() <x)
          {

              prev=current;
              current=current.getLink();
          }
          if (current==head && current.getValue()>x)                  //hook at front
              head=new Node(x,head);
          else if (current.getLink()==null && current.getValue() <x)//hook at end;
          {
              p=new Node(x);
              current.setLink(p);
          }
          else //hook in middle
```

Fig. 9-6b Java LinkedList dass.

```
            {
                p=new Node(x, current);      //set p before current node
                prev.setLink(p);             //hook prev to p
            }
        }
        size++;
    }

    public void Delete(int x)                //removes x from list, does nothing if not
in list
    {
        Node current, prev;
        if (head !=null)                     //list is empty
        {
            //find item
            current=head;
            prev=head;
            while (current.getLink() !=null && current.getValue() !=x)
            {
                prev=current;
                current=current.getLink();
            }
            //if found unhook from list
            if (current==head)
            {
                head=current.getLink();      //unhook from front
                size--;
            }
            else if (current.getValue()==x)
            {
                prev.setLink(current.getLink());//unhook from middle
                size--;
            }
        }                                    //if not found, nothing is done
        }//end delete

        public String toString()             //returns the current list as a string
        {
          Node p;
          String message=" ";
          p=head;
          while (p !=null)
          {
            message=message+p.getValue() +" ";
            p=p.getLink();
          }
          return message;
        }
    }
```


Fig. 9-6b (continued)

The LinkedList class has the same methods as the C++ implementation shown in the previous section with a few minor differences.

Insert(int x): The Insert function maintains the list in order. Methods that insert nodes only at the front or back are shown in the exercises at the end of this chapter.

toString(): The toString() function does not directly print the list, but creates a String for the paint() method of the applet to print. If this were a regular Java application instead of an applet, this method could be coded to do the actual printing.

EXAMPLE 9.6 Write a section of Java code to use the LinkedList class.

Shown below are some consecutive code sections in a paint() function and the output after each g.drawstring() is executed.

Consecutive sections of Code	Output of paint()

`myList=new LinkedList();`
`int yloc=0;`
`myList.Insert(25);`
`myList.Insert(10);`
`myList.Insert(50);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25,yloc+=15);`
	The list is
	10 25 50

`myList.Delete (25);`
`myList.Delete (50);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25, yloc+=15);`
	The list is
	10

`myList.Insert (20);`
`myList.Insert (1);`
`myList.Insert (14);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25, yloc+=15);`
	The list is
	1 10 14 20

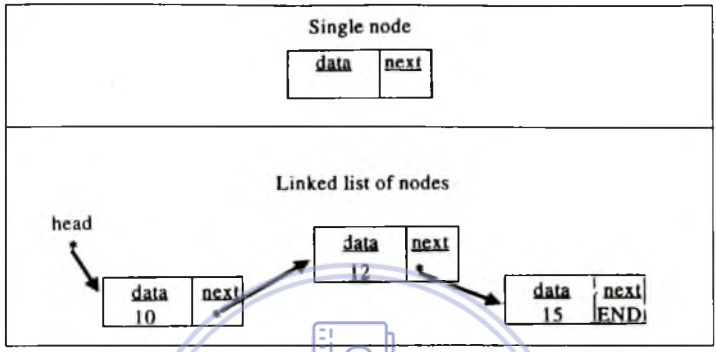
This section has explained the ADT called linked list. Each node in the list contains a field pointing to the location of the next node in the list. The structure can be an ordered or unordered list. Nodes can be inserted in order, at the beginning, or at the end of the list, depending upon the specific application. Other ADTs that can be built using the same linked structure are stacks and queues.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 9.2

9.2 CÁC DANH SÁCH LIÊN KẾT

Kiểu dữ liệu trừu tượng danh sách liên kết (ADT) là một danh sách hoặc một chuỗi các hạng mục ở đó mỗi một hạng mục lại trỏ đến hạng mục kế tiếp trong danh sách. Mỗi hạng mục nằm trong một danh sách liên kết gọi là một nút. Mỗi nút có chứa dữ liệu cần thiết và vị trí của hạng mục kế tiếp. Dữ liệu có thể ở dạng bất kỳ, chẳng hạn như ở một dạng số nguyên, một mảng hoặc thậm chí một đối tượng lớp khác và thường được giữ theo một vài kiểu thứ tự phù thuộc vào ứng dụng chuyên biệt của nó. Nội dung của liên kết sẽ thay đổi đa dạng phụ thuộc vào việc thực thi.

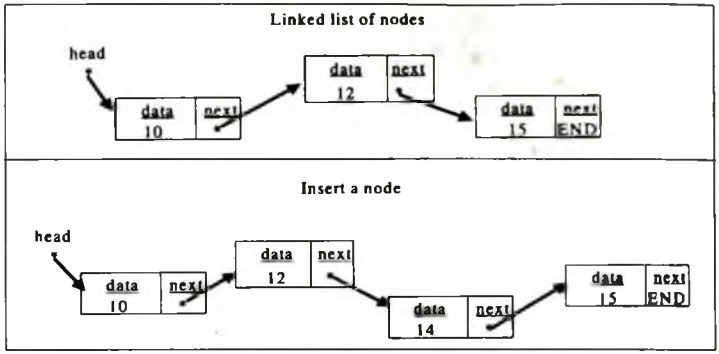
VÍ DỤ 9.1 *Vẽ cách trình bày một nút đơn và một danh sách liên kết. Khảo sát cách trình bày được minh họa trong hình 9.1*



Hình 9.1 *Một nút và một danh sách liên kết*

Danh sách liên kết ADT thường có chứa các biến phần tử dành cho vị trí bắt đầu của danh sách và kích thước của danh sách. Các hàm danh sách liên kết tối thiểu cần thiết để cấu tạo danh sách là: chèn một hạng mục vào danh sách, xóa bỏ một hạng mục khỏi danh sách, và in nội dung của danh sách. Các hàm khác có thể được bổ sung nếu cần chẳng hạn như để báo cáo kích thước của danh sách, in hạng mục thứ k, tìm kiếm danh sách v,v. mã thực tế dành để chèn và hủy bỏ sẽ biến thiên tùy theo danh sách để bảo quản theo thứ tự. Nếu danh sách nằm theo thứ tự, một hạng mục được chèn ở một địa chỉ hoàn chỉnh. Tất cả điều cần làm đó là tạo một nút rồi chỉnh sửa các con trỏ như minh họa ở hình 9.2.

Các danh sách liên kết có một ưu điểm nổi bật so với các mảng đơn. Lúc chèn hoặc xóa bỏ một hạng mục trong một danh sách liên kết có thứ tự, chỉ có các con trỏ là được điều chỉnh. Trong một mảng, tất cả các hạng mục nằm sau điểm chèn đều phải được di chuyển lên hoặc xuống để giữ cho thứ tự hoàn chỉnh. Tuy nhiên trong một mảng, người ta có thể nhìn thấy bất kỳ hạng mục chuyên biệt nào một cách trực tiếp (ví dụ arrayName[31]). Để tìm hạng mục thứ k trong một danh sách liên kết, danh sách liên kết phải được chuyển vị bởi con trỏ theo sau của mỗi một nút đến vị trí của nút kế tiếp. Ta không thể tìm được một hạng mục đơn mà không chuyển vị toàn bộ danh sách. Do đó các mảng là phương thức nhanh để mọt hạng mục đặc biệt và các danh sách liên kết hiệu quả hơn nhiều trong công việc chèn và hủy. Việc thực thi chuyên biệt hầu hết các cấu trúc dữ liệu sẽ cho ta kết quả có một vài kiểu thuận lợi giữa không gian và thời gian.



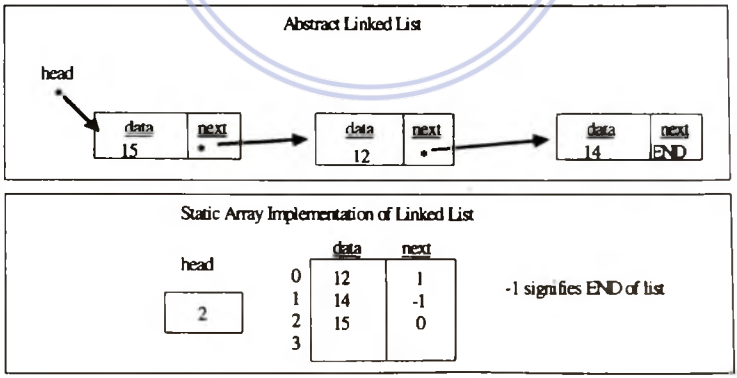
Hình 9.2 *Chèn một nút vào danh sách liên kết*

9.2.1 Thực thi tĩnh của danh sách liên kết

Kỹ thuật để thực thi một danh sách liên kết tĩnh bao gồm việc sử dụng một mảng dành cho không gian danh sách. Ví dụ được dùng trong mục này là một danh sách không được sắp xếp theo thứ tự với các hạng mục được chèn ở cuối danh sách. Mỗi hạng mục trong một mảng là một nút. Liên kết của mỗi một nút có chứa vị trí của hạng mục kế tiếp trong danh sách.

VÍ DỤ 9.2 *Vẽ sơ đồ trình bày một danh sách liên kết tĩnh*

Hình 9.3 minh họa ví dụ này



Hình 9.3 *Danh sách liên kết tĩnh*

Lưu ý hai điều về việc thực thi này. Trước tiên, vùng không gian mảng phải là một vùng các nút có sẵn vốn cũng là một danh sách liên

kết. Lúc chèn một hạng mục, nút có sẵn đầu tiên phải được chọn. Trong hình 9.3, nếu một hạng mục khác được chèn vào danh sách thì nó sẽ được đặt ở điểm 3. Danh sách cần phải giữ lộ trình của nút có sẵn kế tiếp. Thứ hai, mảng có chứa một số hữu hạn các vị trí và phải đủ lớn dành cho tất cả các danh sách có thể có. Ví dụ một mảng có thể được dùng cho danh sách khóa học. Lớp học lớn nhất có 100 học viên và lớp học nhỏ nhất có 20 học viên. Các học viên được bổ sung thêm và được loại bỏ các danh sách. Không gian mảng đã cho có thể lên đến 200 vị trí. Do đó, phải có ba số phải được theo dõi đó là giá trị cực đại có thể có sẵn trong mảng, giá trị cực đại này được dùng trong bất cứ danh sách đã cho nào và chỉ số của số hiện tại trong danh sách.

***VÍ DỤ 9.3** Thực thi trong C++ một danh sách liên kết tĩnh các số nguyên*

*Điều này được thực thi trong C++ thông qua việc sử dụng lớp LinkedList. File LinkedList.h được minh họa ở hình 9.4a. Tất cả các biến phần tử và các hàm cần cho danh sách này cũng được đưa vào. các hàm được thực thi trong hình 9.4b và được giải thích dưới đây. ** Hình 9.4a LinkedList.h.*


```
/*
 *class LinkedList
 */

typedef int Item;
const int MAX = 8;              //maximum possible in the array
const int END = -1;             //constant signifies the end of any list

class Node
{
  public:
    Item value;
    int link;
};

class LinkedList
{
  private:
    Node space[MAX];
    int head;       //address of the first node in list
    int avail;      //address of first node available to be used
    int maxSize;    //maximum number of items in the current list
    int size;       //number of items in current list

public:

    LinkedList(int n);          //Initializes the LinkedList as empty with maximum size n.

    int Size() {return size;}; //returns size of list - use of inline function

    void Insert(Item x);        //Inserts an item x in first available spot

    void Delete(Item x);        //Removes the item x - Prints error if x was not in the list

    void PrintList(char mode); //prints the list in two ways for testing - either
                                //mode=(o) list in order or (a) print array contents
};
```

***Hình 9.4b** LinkedList.cpp.*

```
//LinkedList.cpp - implementation of LinkedList class

#include<iostream.h>
#include "LinkedList.h"
LinkedList::LinkedList(int n)
//Initializes the LinkedList as empty with maximum size n.
{
    head=END;
    size=0;
    avail=0;
    maxSize=n;
    for (int i=0; i<MAX; i++)        //set up all the spaces
    {    space[i].value=END;
         space[i].link=i+1;
    }

    space[MAX-1].link=END;           //end the total list
    space[n-1].link=END;             //end the list of available nodes for the current list
}
void LinkedList::Insert(Item x)
//insert x in first available spot and hook at end of list
{
    int p, current;
    //get the first spot
  if (avail==END) cout<<"\n LinkedList::Insert -- list is full \n";
  else
  {
    p=avail;
    avail=space[avail].link;         //next available spot
    space[p].value=x;                //fill values
    space[p].link=END;
    //find the end to hook it up
    if (head==END) //hook it at the beginning
        head=p;
    else
    {
        current=head;
        while (space[current].link !=END)
            current=space[current].link;
        //now current points to last item
        //hook it up
        space[current].link=p;
    }//end else hook at end
    size++;
  }//end else list is full
}
    void LinkedList::Delete(Item x)
    {
        //find the item
      int prev, current;
      if (head==END) cout<< "\n LinkedList::Delete - List is empty \n";
      else
      {
```

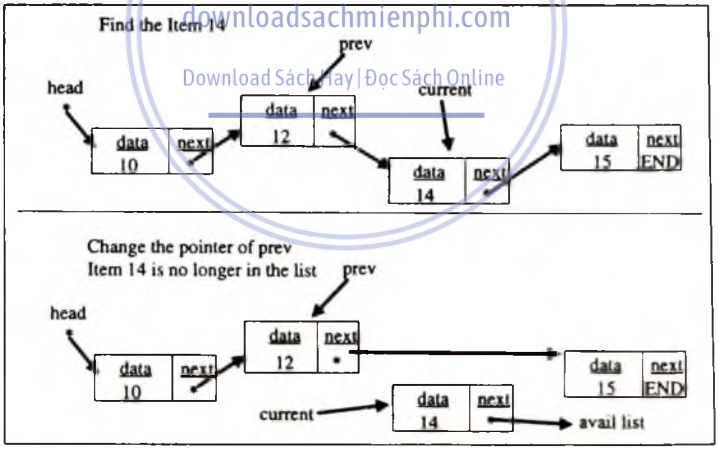
Hình 9.4b *(tiếp theo)*

```
    prev=head;
    current=head;
    while (space[current].link !=END && space[current].value !=x)
    {
        prev=current;
        current=space[current].link;
    }
    if (space[current].value==x) //item is found
    {
        //unhook it
        //check to see if it is first in list
        if (current==head) head=space[current].link;
        else space[prev].link=space[current].link;

        //return space to avail
         space[current].link=avail;
         avail=current;
         size--;
    }
    else cout<< '\n LinkedList::Delete '<<x<<' not found \n';
}//end else list is not empty
 }

 void LinkedList::PrintList(char mode)
 //prints the list either mode=(o) list in order or (a) print array contents
 {
    int i=0;
    switch (mode)
    {
    case 'o':
        cout<<'\nTHE LIST:\n';
        i=head;
        if (i==END) cout<<'Empty\n';
        else
            while (i !=END)
            {
                cout<<space[i].value<<endl;
                i=space[i].link;
            }
        break;
    case 'a':
        cout<<'\nTHE LIST: starts in spot '<<head<<endl;
        cout<<'The first available spot is '<<avail<<endl;
        for (i=0; i<MAX; i++)
            cout<<'['<<i<<']'<<'\t'<<space[i].value<<'\t'<<space[i].link<<endl;
        break;
    default:
        cout<<'LinkedList: PrintList - error in mode\n';
    }
    cout<<endl;
 }
```


***Hình 9.4b** (tiếp theo)*

LinkedList(int n): *Bộ cấu tạo xác lập các hạng mục cần thiết dành cho một danh sách trống để có thể chỉ phát triển sang một kích thước n. Đầu của danh sách được xác lập sang - 1 để chỉ rõ một danh sách trống. Vị trí có sẵn đầu tiên là vị trí 0, và danh sách các nút có sẵn tất cả đều trỏ đến vị trí trống theo sau đó. Đối với mỗi nút, giá trị là tùy ý và được khởi tạo sang END. (Sự khởi tạo có thể được lập sang 0, hoặc 999, hoặc bất cứ số nào khác được chọn để trình bày một giá trị không xác định.) sau cùng, hạng mục cuối cùng trong mảng và hạng mục cuối cùng trong danh sách có sẵn cả hai đều được xác lập sang END.*

Insert(Itemx): *hàm Insert kiểm tra xem thử danh sách có đầy hay không (cần thiết cho các cấu trúc tĩnh của kích thước hữu hạn), nhận điểm có sẵn kế tiếp, điều chỉnh danh sách biến rồi bổ sung hạng mục ở cuối danh sách. Một thuật toán khác có thể được viết để chèn hạng mục ở đầu. Cũng vậy nếu danh sách được xếp theo thứ tự, thì một thuật toán được viết để chèn hạng mục này vào vị trí tiêu biểu của nó. Các hàm này được minh họa trong bài tập ở cuối chương. Bởi vì câu lệnh này theo dõi kích thước của danh sách hiện tại, cho nên kích thước biến này được gia tăng.*



Hình 9.5 *Xóa bỏ một hạng mục khỏi danh sách - abstract view*

Delete(Item x): *Hàm Delete trước tiên kiểm tra xem thử danh sách có bị trống hay không. Nếu danh sách này không trống thì hàm sẽ tìm kiếm hạng mục bị xóa bỏ. Cần phải có hai con trỏ bên ngoài: một để*

trỏ đến nút hiện tại và một để trỏ đến nút trước đó. Con trỏ trước đó phải đi theo sau con trỏ hiện tại thông qua danh sách để lúc hạng mục được tìm thấy, thì nó có thể bị xóa bỏ thông qua việc điều chỉnh hai con trỏ. Xem hình 9.5 để có một ví dụ. Mỗi khi hạng mục đã được tìm thấy, thì hạng mục đó sẽ bị xóa bỏ khỏi danh sách và vị trí trả về danh sách các nút có sẵn để có thể được dừng lại.

PrintList(char mode): *Thông thường hàm Print sẽ in nội dung của danh sách theo thứ tự. Tuy nhiên, hàm Print in danh sách theo hai cách vì mục đích giải thích và vì thử nghiệm chương trình.Lúc chế độ nằm ở vị trí "0" thì các giá trị của danh sách được in theo thứ tự mà chúng xảy ra trong danh sách. Với chế độ ở vị trí "a" thì toàn bộ mảng được in. Hàm print này có thể điều phối để in bất cứ cách nào cần thiết bởi một trình ứng dụng đặc biệt.*

VÍ DỤ 9.4 *Viết một mục mã C++ để sử dụng cấu trúc LinkedList.*

Dưới đây là phần minh họa các mục mã liên tục trong hàm main() và hình ảnh của danh sách và mảng sau mỗi một mục.

Code	mylist.PrintList('o');	mylist.PrintList('a');

`LinkedList mylist(5);`
 	
//sets up empty list with:
//MAX of array = 8 (see .h file)
//max size of this list = 5
	The list:
	Empty
		The list: starts in spot −1
		The first available spot is 0
		[0] −1 1
		[1] −1 2
		[2] −1 3
		[3] −1 4
		[4] −1 −1
		[5] −1 6
		[6] −1 7
		[7] −1 −1

`mylist.Insert(10);`
`mylist.Insert(14);`
`mylist.Insert(9);`
`mylist.Insert(3);`
`mylist.Insert(15);`
 	
//list is now full, so avail = −1
	The list:
	10
	14
	9
	3
	15
		The list: starts in spot 0
		The first available spot is −1
		[0] 10 1
		[1] 14 2
		[2] 9 3
		[3] 3 4
		[4] 15 −1
		[5] −1 6
		[6] −1 7
		[7] −1 −1

`mylist.Delete(10);`
`mylist.Delete(3);`
 	
/* Start with 1 and follow the links for the list.
	The list:
	14
	9
	15
		The list: starts in spot 1
		The first available spot is 3
		[0] 10 −1
		[1] 14 2
		[2] 9 4

Start with 3 and follow the links for the available nodes. Remember, only 0 through 4 can be used*/ `myList.Insert(25);` /* Again, start with 1 and follow the links for the list. Start with 3 and follow the links for the available nodes.*/	The list: 14 9 15 25	[3] 3 0 [4] 15 −1 [5] −1 6 [6] −1 7 [7] −1 −1 The list: starts in spot 1 The first available spot is 0 [0] 10 −1 [1] 14 2 [2] 9 4 [3] 25 −1 [4] 15 3 [5] −1 6 [6] −1 7 [7] −1 −1

9.2.2 Thực thi động danh sách liên kết

Thực thi tĩnh của danh sách liên kết trên đây có hai vấn đề. Trước tiên, mảng phải được tạo ra đủ lớn cho bất cứ ứng dụng nào có thể có, thậm chí, ngay cả khi không có khoảng trống nào được dùng trong danh sách đã cho. Cũng như các nút có sẵn trong danh sách phải được giữ lại bởi lớp đó. Trong việc thực thi động, không gian có sẵn tất cả là bộ nhớ gọi là heap space, và các vị trí có sẵn được theo dõi bởi máy tính, các nút được tạo theo nhu cầu và trả về không gian heap để sử dụng sau này. Lớp danh sách chỉ cần một con trỏ trỏ đến đầu của danh sách và một số nguyên theo dõi kích thước.

VÍ DỤ 9.5 *Hãy viết một mã Java để thực thi danh sách liên kết theo thứ tự.*

```
//class definition (Node.java file)

public class Node
{
    int value;
    Node link;                      //pointer to itself is implicit in Java

    Node (int x) this(x, null);}  //constructor to create a node with data of x

    Node(int x, Node next) //constructor creates node with data of x and points to next in list
    {
      value=x;
      link=next;
    }

    int getValue() {return value;}                  //return the value of the node

    Node getLink() {return link;}                   //returns the next node in the list

    void setLink(Node newLink) {link=newLink;}  //resets the value of link
}
```

Hình 9.6a *Lớp Java Node.*

Trong một danh sách thứ tự, một nút phải nhỏ hơn hoặc bằng nút theo sau nó. Trong trường hợp này thì Node là một lớp riêng biệt để

xử lý các phép tính của riêng nó. Các hàm này cho phép lớp LinkedList truy cập vào các biến lớp Node. Có hai bộ cấu trúc: một để tạo nên một Node với một liên kết trống (null), và một để tạo ra một Node trỏ đến một nút khác. Hình 9.6 a có chứa lớp Node và hình 9.6b biểu thị lớp LinkedList.

```
//class definition (LinkedList.java file)

public class LinkedList
{
  //instance variables
  private Node head;          //only head of list needed - all memory is available
  private int size;           //current size of list
  //methods
  public LinkedList(){ head=null; }          //default constructor

  int getSize (){ return size; }             //returns size of list

  public void Insert(int x)                  //inserts x and hooks to begining of list
     {    Node p, prev, current;
          if (head==null)                    //list is empty
          head=new Node(x);
          else                               //insert in order
          {
            //find spot to insert
            current=head;
            prev=head;
            while (current.getLink() !=null && current.getValue() <x)
            {
                prev=current;
                current=current.getLink();
            }
            if (current==head && current.getValue()>x)          //hook at front
                head=new Node(x,head);
            else if (current.getLink()==null && current.getValue() <x)//hook at end;
            {
                p=new Node(x);
                current.setLink(p);
            }
            else //hook in middle

            {
                p=new Node(x, current);      //set p before current node
                prev.setLink(p);             //hook prev to p
            }
        }
        size++;
     }

     public void Delete(int x)               //removes x from list, does nothing if not
   in  list
     {
        Node current, prev;
        if (head !=null)                     //list is empty
        {
            //find item
            current=head;
            prev=head;
            while (current.getLink() !=null && current.getValue() !=x)
            {
                prev=current;
                current=current.getLink();
            }
            //if found unhook from list
            if (current==head)
            {
                head=current.getLink();      //unhook from front
                size--;
            }
```

***Hình 9.6b** Lớp Java LinkedList*

```
        else if (current.getValue()==x)
        {
            prev.setLink(current.getLink());unhook from middle
            size--;
        }
    }                                   //if not found, nothing is done
    }//end delete

    public String toString()            //returns the current list as a string
    {
      Node p;
      String message=" ";
      p=head;
      while (p !=null)
      {
        message=message+p.getValue() +" ";
        p=p.getLink();
      }
      return message;
    }
}
```

***Hình 9.6b** (tiếp theo)*

Lớp LinkedList có các phương pháp giống hệt như C++ như minh họa ở phần trước đây với một vài sự khác biệt nhỏ.

***Insert(int x):** Hàm Insert giữ lại danh sách theo thứ tự. Các phương pháp để chèn các nút chỉ ở đằng trước hoặc đằng sau như minh họa trong các bài tập ở cuối chương này.*

***ToString():**Hàm toString() không in trực tiếp danh sách , nhưng tạo ra một string (chuỗi) dành cho phương pháp paint() của trình ứng dụng để in. Nếu đây là một trình ứng dụng Java bình thường thay vì một applet, thì phương pháp này có thể được tạo mã để thực hiện việc in ấn thực tế.*

***VÍ DỤ 9.6** Hãy viết một phần trong mã Java để sử dụng lớp LinkedList.*

Phần minh họa dưới đây là một vài mục mã kế cận nhau trong hàm paint() và kết quả xuất sau khi g.drawstring() được thực thi.

Consecutive sections of Code	Output of paint()

`myList=new LinkedList();`
`int yloc=0;`
`myList.Insert(25);`
`myList.Insert(10);`
`myList.Insert(50);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25,yloc+=15);`
	The list is
	10 25 50

`myList.Delete (25);`
`myList.Delete (50);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25, yloc+=15);`
	The list is
	10

```
myList.Insert (20);
myList.Insert (1);
myList.Insert (14);
g.drawString ("The list is ",25, yloc+=15);
g.drawString (myList.toString(), 25, yloc+=15);
```

The list is
1 10 14 20

Mục này đã giải thích ADT được gọi là danh sách liên kết.Mỗi nút trong danh sách này có chứa một trường trỏ đến vị trí nút kế tiếp của danh sách. Cấu trúc có thể là một danh sách theo thứ tự hoặc một danh sách không theo thứ tự. Các nút có thể được chèn theo thứ tự, ở phần đầu hoặc ở phần cuối của danh sách, phụ thuộc vào các ứng dụng chuyên biệt. Các ADT khác có thể được cấu tạo bằng cách sử dụng cấu trúc liên kết giống như các hàng chồng và các hàng đợi.

CHỦ ĐIỂM 9.3

STACKS
Xếp chồng (Hàng chồng)

A **stack** is a particular type of linked list where the items are always added to the top and removed from the top. The stack is usually called a last-in, first-out (LIFO) structure. One can have a stack of cards, a stack of plates, or a stack of clothing. In each case, the only available item is the one on the top. Many applications, such as the parsing of input data, rely on stacks. The computer also uses stacks during the processing of subroutine calls to keep track of the calling sequence.

EXAMPLE 9.7 Write a C++ implementation of a stack.

A C++ Stack class is shown below. The Stack.h interface file is in Fig. 9-7a and the Stack.cpp implementation is in Fig. 9-7b. This stack implements dynamic memory and a stack of characters rather than the integers shown in the linked lists of the previous section. In C++, a pointer variable is declared using an asterisk (e.g., Node *p;) and then the fields of the node are accessed through the "->" operator (e.g., p->value = 'a';) instead of through the dot operator shown above in Java.


```
//Stack.h - dynamic version of stack
#include <iostream.h>
typedef char Item;

class Node

{
public:
    Item value;
    Node *link;
};

class Stack
{
private:
    Node *stackTop; //a pointer to the top Node of the stack (or NULL if empty)

public:
    Stack();              //initializes a new stack as empty
    void Push(Item x);    //pushes the item x onto the stack

    Item Pop();           //pops and returns the top item from the stack

    Item Top();           //returns the top item from the stack without popping it

    int Size();           //returns the number of items in the stack

    void PrintStack();    //prints the contents of the stack

};
```

Fig. 9-7a Stack.h.

The only private member variable in this example is the pointer to the top of the stack. The member functions in addition to the constructor are:

- push, or insert, an item onto the top of the stack
- pop, or remove, the top item
- look at the top item but leave it on the stack
- find the size
- print the stack

Push(Item x): It is important to note that in the push function, the only time the stack is full is if the computer's entire memory is used and the new() call returns a NULL. This can occasionally happen if a.push function is in the body of an infinite loop. If the stack is not full, the Item x is placed at the top of the stack.

```
//Stack.cpp - dynamic version of stack
#include<iostream.h>
#include "Stack.h"
Stack::Stack() { stackTop=NULL; }          //initializes a new stack as empty
void Stack::Push(Item x)                   //pushes the item x onto the stack
{
    Node *p;
    p=new Node;
    if (p==NULL) cout<< "\n Stack::Push - entire memory is full \n";
    else
    {
        p->value=x;
        p->link=stackTop;
        stackTop=p;
    }
}
Item Stack::Pop()                          //pops and returns the top item from the stack
{
    Item x=0;                              //creates a null character
    Node *p;
    if (stackTop==NULL) cout<< "\n Stack::Pop - empty stack\n";
    else
    {
        p=stackTop;
        x=p->value;
        stackTop=p->link;
        delete p;                          //return location to available memory
    }
    return x;
}
Item Stack::Top()          //returns the top item from the stack without popping it
{
    Item x=0;              //creates a null character
    if (stackTop==NULL) cout<< "\n Stack::Top - empty stack \n";
    else x=stackTop->value;
    return x;
}
int Stack::Size()          //returns the number of items in the stack
{
    int j=0; Node *p;
    p=stackTop;
    while (p !=NULL)
```


Fig. 9-7b Stack.cpp.

```
    while (p !=NULL)
    {
        p=p->link;
        j++;
    }
    return j;
}
void Stack::PrintStack() //prints the contents of the stack
{
    Node *p;
    p=stackTop;
    cout<<'\nThe STACK: \n';
    while (p !=NULL)
    {
        cout<<p->value<<endl;
        p=p->link;
    }
}
```

Fig. 9-7b (Continued)

Pop(): The pop adjusts the *stackTop* pointer to remove the first item in the stack if the stack is not empty. The memory is returned to the operating system through the *delete p command*, and the Item is returned as the function value to the calling procedure.

TopQ: This function returns the first item without actually removing it from the stack. In some implementations this is called *peek*.

Size() and PrintStackQ: Both these functions traverse the entire stack, one counting the items and the other printing them. A separate pointer p must be used to traverse the stack rather than the *stackTop so* that the stack remains intact.

EXAMPLE 9.8 Write a short section of code to create and manipulate a simple character stack. Here is a sample main() function along with the resultant output.

Consecutive Code Sections	Output

`Item x;`
`Stack myS;`
`myS.Push('c');`
`myS.Push('a');`
`myS.Push('t');`
`myS.PrintStack();`
	The STACK:
	t
	a
	c

`myS.Push('s');` `myS.PrintStack();`	The STACK: s t a c

`x=myS.Pop();`
`cout<<endl<<x<<' is popped\n';`
`myS.PrintStack();`
	s is popped
	The STACK:
	t
	a
	c

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 9.3

9.3 HÀNG CHỒNG

Hàng chồng là một kiểu danh sách liên kết đặc biệt ở đó các hạng mục luôn luôn được cộng vào trên đầu và bị xóa bỏ đầu. Các hàng chồng thường được gọi là cấu trúc Last-In, First-Out (LIFO) (vào sau ra trước). Người ta có thể có một chồng các lá bài, một chồng các tấm hoặc một chồng áo quần. Trong mỗi một trường hợp hạng mục có sẵn duy nhất chính là hạng mục bên trên cùng. Nhiều trình ứng dụng chẳng hạn như việc xử lý rời rạc dữ liệu đặt cơ sở trên các hàng chồng. Máy tính cũng sử dụng các hàng chồng trong quá trình xử lý các cuộc gọi theo thủ tục con để theo dõi tuần tự gọi.

***VÍ DỤ 9.7** Hãy viết một C++ thực thi một hàng chồng.*

*Lớp C++ Stack được minh họa dưới đây. File giao diện Stack.h ở hình 9.7a và file thực thi Stack.cpp được cho ở hình 9.7b. Chồng này thực thi bộ nhớ động và một hàng chồng các ký tự thay vì các số nguyên như được minh họa trong các danh sách liên kết của mục trước đây. Trong C++, một biến con trỏ được khai báo bằng cách sử dụng một ký tự thay thế (asterisk) (ví dụ, Node*p;) và rồi các trường của nút được thông qua toán tử "->" (ví dụ, p->value = "a";) thay vì thông qua một toán tử chấm như minh họa trên đây trong Java.*

```
//Stack.h - dynamic version of stack

#include <iostream.h>
typedef char Item;

class Node

{
public:
    Item value;
    Node *link;
};

class Stack
{
private:
    Node *stackTop; //a pointer to the top Node of the stack (or NULL if empty)

public:
    Stack();              //initializes a new stack as empty
    void Push(Item x);    //pushes the item x onto the stack

    Item Pop();           //pops and returns the top item from the stack

    Item Top();           //returns the top item from the stack without popping it

    int Size();           //returns the number of items in the stack

    void PrintStack();    //prints the contents of the stack

};
```

***Hình 9.7a** Stack.h*

```
//Stack.cpp - dynamic version of stack

#include<iostream.h>
#include "Stack.h"

Stack::Stack() { stackTop=NULL; }          //initializes a new stack as empty

void Stack::Push(Item x)                   //pushes the item x onto the stack
{
    Node *p;
    p=new Node;
    if (p==NULL) cout<< "\n Stack::Push - entire memory is full \n";
    else
    {
        p->value=x;
        p->link=stackTop;
        stackTop=p;
    }
}

Item Stack::Pop()                          //pops and returns the top item from the stack
{
    Item x=0;                              //creates a null character
    Node *p;
    if (stackTop==NULL) cout<< "\n Stack::Pop - empty stack\n";
    else
    {
        p=stackTop;
        x=p->value;
        stackTop=p->link;
        delete p;                          //return location to available memory
    }
    return x;
}

Item Stack::Top()          //returns the top item from the stack without popping it
{
    Item x=0;              //creates a null character
    if (stackTop==NULL) cout<< "\n Stack::Top - empty stack \n";
    else x=stackTop->value;
    return x;
}

int Stack::Size()          //returns the number of items in the stack
{
    int j=0; Node *p;
    p=stackTop;
    while (p !=NULL)
    {
        p=p->link;
        j++;
    }
    return j;
}

void Stack::PrintStack() //prints the contents of the stack
{
    Node *p;
    p=stackTop;
    cout<<"\nThe STACK: \n";
    while (p !=NULL)
    {
        cout<<p->value<<endl;
        p=p->link;
    }
}
```


Hình 9.7b Stack.cpp.

Phần tử riêng biệt duy nhất có sẵn trong ví dụ này là con trỏ trỏ đến trên cùng của chồng. Các hàm phần tử bổ sung cho bộ cấu trúc là:

- *đẩy hoặc chèn một hạng mục lên phần cao nhất của hàng chồng*
- *di dời hạng mục trên cùng*
- *xem xét hạng mục trên cùng nhưng vẫn để nó ở trên hàng chồng*
- *tìm kích thước*
- *in hàng chồng*

Push(Item x): *Điều quan trọng cần lưu ý rằng trong hàm push, thời điểm duy nhất để hàng chồng được đầy đó là nếu bộ nhớ tổng thể máy tính được dùng và cuộc gọi new() trả về một NULL. Điều này có thể xảy ra thỉnh thoảng nếu hàm push nằm trongvòng lặp vô hạn. Nếu hàng chồng này không bị đầy, thì hạng mục x được đặt ở trên đầu của chồng.*

Pop(): *Pop điều chỉnh con trỏ để xóa bỏ hạng mục đầu tiên trong hàng chồng nếu hàng chồng này không trống. Bộ nhớ này được trả về với hệ điều hành thông qua lệnh delete p, và Item được trả về dưới dạng một giá trị hàm cho việc gọi thủ tục.*

Top(): *Hàm này trả về hạng mục đầu tiên mà không thật sự xóa bỏ nó khỏi hàng chồng. Trong một vài quy trình thực thi nó được gọi là peek.*

Size()and PrintStack(): *Cả hai hàm này đều chuyển vị toàn bộ hàng chồng, một hàm để đếm các hạng mục và kia in chúng. Một con trỏ p riêng biệt phải được dùng để chuyển vị hàng chồng thay vì dùng stack Top để hàng chồng này vẫn còn nguyên vẹn.*

VÍ DỤ 9.8 *Viết một đoạn mã ngắn để tạo và xử lý một chồng ký tự đơn giản. Dưới đây là ví dụ về hàm main() cùng với kết quả xuất của nó.*

Consecutive Code Sections	Output

`Item x;`
`Stack myS;`
`myS.Push('c');`
`myS.Push('a');`
`myS.Push('t');`
`myS.PrintStack();`
	The STACK:
	t
	a
	c

`myS.Push('s');` `myS.PrintStack();`	The STACK: s t a c

`x=myS.Pop();`
`cout<<endl<<x<<' is popped\n';`
`myS.PrintStack();`
	s is popped
	The STACK:
	t
	a
	c

CHỦ ĐIỂM 9.4

QUEUES
Hàng đợi

A queue is a particular type of linked list where the items are always added to the back and removed from the front. The queue is usually called a first-in, first-out (FIFO) structure. There can be a queue of people purchasing tickets, a queue of cars waiting at a car wash, or a queue of items on a conveyor belt. In each case, items are added at the back and leave from the front. Network operating systems use queues to process printing requests. Documents can be submitted to a printer, and the printer will process these documents in the order in which they were received.

EXAMPLE 9.9 Write the Java code to implement a Queue class. A Java Queue class is shown below. The Node.Java is in Fig. 9-8a and the Queue.Java is in Fig. 9-8b. This queue implements dynamic memory with a queue of characters rather than the integers shown in the linked lists.


```
//Node.java for use in character Queue

public class Node
{
    char value;
    Node link; //pointer is implicit in Java

    //constructor to create a node with data of s
    Node (char x)     {this(x, null);}

    //constructor creates node with data of x and points to next in list
    Node(char x, Node next)
    {
      value=x;
      link=next;
    }

    //returns the value of the node
    char getValue() {return value;}

    //returns the next node in the list
    Node getLink() {return link;}

    //resets the value of link
    void setLink(Node newLink) {link=newLink;}
}
```

Fig. 9-8a Node.Java.

Instance variables are pointers to the front and the rear of the queue. The methods implemented are similar to the ones needed for the Stack: the constructor, putting something onto the rear of the queue, removing a

node from the front of the queue, looking at the front of the queue, finding the size, and returning the string value of the queue. Logic for these Queue functions is similar to the logic of the Stack functions.

```
//class definition (Queue.java file)
public class Queue
{
   //instance variables
   private Node front, rear;              //pointer to front and rear

   //methods
   public Queue()                         //default constructor
   { front=rear=null; }

   public void Enqueue(char x)            //inserts x at the end of the queue
   {
     Node p=new Node(x);                  //create node with null link;
     if (rear==null)                      //if list is empty, item is first
     {
        front=p;
        rear=front;
     }
     else                                 //insert at rear
     {
        rear.setLink(p);
        rear=p;
     }
   }

   public char Dequeue()                  //removes from front of list
   {
     char ch=' ';
     if (front !=null)                    //if queue not empty
     {
       ch=front.getValue();
       front=front.getLink();             //unhook from front
     }
     return ch;
   }//end delete

   public char Front()                    //returns front of list
   {
     char ch=' ';
     if (front !=null)                    //if queue not empty
       ch=front.getValue();
     return ch;
   }//end delete
```


Fig. 9-8b Queue.java

```
    int Size ()                          //returns size of queue
    {
      Node p;
      int j=0;
      p=front;
      while (p !=null)
      {
        j++;
        p=p.getLink();
      }
      return j;
    }
    public String toString()             //returns the current queue as a string
    {
      Node p;
      String message=" ";
      p=front;
      while (p !=null)
      {
        message=message+p.getValue()+" ";
        p=p.getLink();
      }
      return message;
    }
}
```


Fig. 9-8b (Continued)

EXAMPLE 9.10 Write a short Java function to test the Queue class.

Here is a sample ***paint()*** function to create and manipulate a simple character queue, along with the resultant output.

Consecutive Code Sections	Output

`myq=new Queue();`
`int yloc=0;`
`for (char ch='A'; ch<'D'; ch++) myq.Enqueue(ch);`
`g.drawString ("The queue is ",25, yloc+=15);`
`g.drawString (myq.toString(), 25,yloc+=15);`
	The queue is
	A B C

`char x=myq.Dequeue();`
`g.drawString`
`  (x+" was the character dequeued ",25, yloc += 25);`
`g.drawString ("The queue is ",25, yloc+=25);`
`g.drawString (myq.toString(), 25, yloc+=15);`
	A was the character dequeued
	The queue is
	B C

`for (char ch='x'; ch<='z'; ch++) myq.Enqueue(ch);`
`g.drawString`
`  ("The front of the q is "+myq.Front(),25, yloc+=25);`
`g.drawString ("The queue is ",25, yloc+=25);`
`g.drawString (myq.toString(), 25, yloc+=15);`
	The front of the q is B
	The queue is
	B C x y z

Many built-in structures are available for use both in C++ and Java. Other ADTs can be built by the programmer and adapted for any application. The linked lists, stacks, and queues explained in this chapter provide a starting-point. In each case, the Node contains the actual data. Those data could be an integer, a character, a string, or an object of any other class. Recall that one of the goals of object-oriented programming is the reuse of code. The sequential development of the structures in this chapter effectively illustrates this point. Several of the exercises at the end provide the reader a way to extend the use of these data structures.

HƯỚNG DẪN ĐỌC HIỂU CHỦ ĐIỂM 9.4

9.4 HÀNG ĐỢI

Một hàng đợi chính là một kiểu danh sách liên kết đặc biệt ở các hạng mục luôn luôn được bổ sung vào đuôi và bị xóa bỏ khỏi đầu. Hàng đợi đuôi này còn được gọi là cấu trúc First-In, First-Out (FIFO) (vào trước ra trước). Ở đây có thể là xếp hàng đuôi những người đang mua vé, một xếp hàng các xe hơi chờ đợi để sửa xe, một hàng đợi các hạng mục trong một dây chuyềnchuyền tải. Trong mỗi trường hợp như thế các hạng mục được bổ sung vào cuối và ra khỏi đầu. Các hệ điều hành mạng sử dụng các hàng đợi để xử lý các yêu cầu in ấn. Các tài liệu có thể truyền vào trong một máy in, và máy in sẽ xử lý những tài liệu này theo thứ tự mà chúng nhận được.

VÍ DỤ 9.9 *Hãy viết một mã Java để thực thi lớp Queue. Một lớp Java Queue được minh họa dưới đây. Node.Java nằm ở hình 9.8a và Queue.java nằm ở hình 9.8b. Hàng đợi này thực thi bộ nhớ động với một đuôi các ký tự thay vì một số nguyên được minh họa trong các danh sách.*

```
//Node.java for use in character Queue

public class Node
{
    char value;
    Node link; //pointer is implicit in Java

    //constructor to create a node with data of s
    Node (char x)     {this(x, null);}

    //constructor creates node with data of x and points to next in list
    Node(char x, Node next)
    {
      value=x;
      link=next;
    }
```

Hình 9.8a *Node.java*

```
    //returns the value of the node
    char getValue() {return value;}

    //returns the next node in the list
    Node getLink() {return link;}

    //resets the value of link
    void setLink(Node newLink) {link=newLink;}
}
```

***Hình 9.8a** (tiếp theo)*

Các biến trường hợp (instance variable) là các con trỏ trỏ đến đầu và đuôi của hàng. Các phương pháp được thực thi thì tương tự như những phương pháp cần thiết cho các hàng chồng: bộ cấu tạo, đặt một vài nội dung vào đuôi của hàng đợi, xóa bỏ một nút khỏi đầu của hàng, xem xét đuôi của hàng tìm kích thước, và trả về giá trị chuỗi của hàng. Logic của các hàm Queue tương tự như logic của các hàm Stack.

```
//class definition (Queue.java file)
public class Queue
{
   //instance variables
   private Node front, rear;              //pointer to front and rear

   //methods
   public Queue()                         //default constructor
   { front=rear=null; }

   public void Enqueue(char x)            //inserts x at the end of the queue
   {
     Node p=new Node(x);                  //create node with null link;
     if (rear==null)                      //if list is empty, item is first
     {
        front=p;
        rear=front;
     }
     else                                 //insert at rear
     {
        rear.setLink(p);
        rear=p;
     }
   }

   public char Dequeue()                  //removes from front of list
   {
     char ch=' ';
```

***Hình 9.8b** Queue.java*

```
    if (front !=null)                    //if queue not empty
    {
      ch=front.getValue();
      front=front.getLink();             //unhook from front
    }
    return ch;
  }//end delete

  public char Front()                    //returns front of list
  {
    char ch=' ';
    if (front !=null)                    //if queue not empty
      ch=front.getValue();
    return ch;
  }//end delete

  int Size ()                            //returns size of queue
  {
    Node p;
    int j=0;
    p=front;
    while (p !=null)
    {
      j++;
      p=p.getLink();
    }
    return j;
  }
  public String toString()               //returns the current queue as a string
  {
    Node p;
    String message=" ";
    p=front;
    while (p !=null)
    {
      message=message+p.getValue()+" ";
      p=p.getLink();
    }
    return message;
  }
}
```


***Hình 9.8b** (tiếp theo)*

***VÍ DỤ 9.10** Viết một hàm Java ngắn để thử nghiệm lớp Queue.*

Dưới đây là một hàm paint() mẫu để tạo và xử lý một hàng ký tự đơn giản, cùng với kết quả xuất của nó.

Consecutive Code Sections	Output

`myq=new Queue();`
`int yloc=0;`
`for (char ch='A'; ch<'D'; ch++) myq.Enqueue(ch);`
`g.drawString ("The queue is ",25, yloc+=15);`
`g.drawString (myq.toString(), 25,yloc+=15);`
	The queue is
	A B C

`char x=myq.Dequeue();`
`g.drawString`
`(x+" was the character dequeued ",25, yloc += 25);`
`g.drawString ("The queue is ",25, yloc+=25);`
`g.drawString (myq.toString(), 25, yloc+=15);`
	A was the character dequeued
	The queue is
	B C

`for (char ch='x'; ch<='z'; ch++) myq.Enqueue(ch);`
`g.drawString`
`("The front of the q is "+myq.Front(),25, yloc+=25);`
`g.drawString ("The queue is ",25, yloc+=25);`
`g.drawString (myq.toString(), 25, yloc+=15);`
	The front of the q is B
	The queue is
	B C x y z

Nhiều cấu trúc được tạo sẵn để sử dụng cho cả C++ và Java. ADT khác có thể được cấu tạo bởi nhà lập trình và được mô phỏng cho bất kỳ trình ứng dụng nào. Các danh sách liên kết, các hàng chồng và các hàng đợi được giải thích trong chương này cung cấp cho bạn một điểm khởi đầu. Trong mỗi một trường hợp, Node có chứa dữ liệu thật sự. Dữ liệu có thể là một số nguyên, một kỹ tự, một chuỗi, hoặc một đối tượng hoặc bất kỳ lớp nào khác. Hãy nhớ lại rằng một trong những mục tiêu của lập trình hướng đối tượng đó là sử dụng lại mã. Sự phát triển tuần tự của các cấu trúc trong chương này minh họa hiệu quả quan điểm này. Nhiều bài tập ở cuối chương cung cấp cho người đọc một cách mở rộng công dụng của các cấu trúc dữ liệu này.

SOLVED PROBLEMS
Bài tập có lời giải

9.1 Which languages used in this book have built-in structures of the following types?

(a) string

(b) arrays

(c) classes

(d) linked lists

(a) Visual Basic and Java; C++ allows character arrays only

(b) Visual Basic, C++ and Java; most languages provide for arrays

(c) C++ and Java; Visual Basic does not allow classes in most early versions

(d) none; these must be created in most languages

9.2 Some ADTs which have been defined in this book are: arrays, linked lists, stacks, and queues. Which of these would be appropriate for the following applications?

(a) Grocery list

(b) Simulation of a grocery store checkout

(c) Company layoff policy where the people who had worked for the shortest period of time are laid off first

(d) University student records

Most appropriate ADP.

(a) Simple array or unordered linked list

(b) Queue

(c) Stack

(d) Ordered linked list or array

9.3 Add a function to the LinkedList class in Fig. 9-4 that would insert the item at the beginning of the list. The function heading would be:

```
void InsertBeginning(Item x);
```

Code:

```
void LinkedList::InsertBeginning(Item x)
//Inserts an item x in first available spot at the begin-
   ning of the list
{
```

```
    int p, current;
    //get the first spot
if (avail==END) cout<<"\n LinkedList::Insert -- list is full \n";
else
{
    p=avail;
    avail=space[avail].link;      //next available spot
    space[p].value=x;             //fill value

    //hook it at the beginning
    space[p].link=head;
    head=p;

    size++;
  }//end else list is full
}
```

This table demonstrates the use of the InsertBeginning(Item x) function:

Code	mylist.PrintList('o');	mylist.PrintList('a');

`LinkedList mylist(5);`
`Mylist.InsertBeginning(10);`
`Mylist.InsertBeginning(14);`
`Mylist.InsertBeginning(9);`
`Mylist.InsertBeginning(3);`
`Mylist.InsertBeginning(15);`
	The list:
	15
	3
	9
	14
	10
		The list: starts in spot 4
		The first available spot is −1
		[0] 10 −1
		[1] 14 0
		[2] 9 1
		[3] 3 2
		[4] 15 3
		[5] −1 6
		[6] −1 7
		[7] −1 −1

9.4 Draw the array for Fig. 9-5, before and after the deletion of 14.

Before	After deleting 14
[0] 10 1 head=0 [1] 12 2 avail=4 [2] 14 3 [3] 15 −1 [4] −1 −1 [5] −1 6 [6] −1 7 [7] −1 −1	[0] 10 1 head=0 [1] 12 3 avail=2 [2] 14 4 [3] 15 −1 [4] −1 −1 [5] −1 6 [6] −1 7 [7] −1 −1

Note: Following the links through the list begins at the head 0, and goes in this order: 0, 1, 3, -1(stop). Following the links through the available list begins at 2 and goes in this order: 2, 4, -1(stop).

9.5 What would be the output of the following sections of code for the unordered static C++ LinkedList class?

Code	mylist.PrintList('o');	mylist.PrintList('a');

LinkedList mylist(6);
 	
`//sets up empty list with:`
`//MAX of array=8 (see .h file)`
`//max size of this list=6`
	The list:
	Empty
		The list: starts in spot −1
		The first available spot is 0
		[0] −1 1
		[1] −1 2
		[2] −1 3
		[3] −1 4
		[4] −1 5
		[5] −1 −1
		[6] −1 7
		[7] −1 −1

`for (int i=0; i<3; i++)` `  mylist.Insert(50+(4*i));`	The list: 50 54 58	The list: starts in spot 0 The first available spot is 3 [0] 50 1 [1] 54 2 [2] 58 −1 [3] −1 4 [4] −1 5 [5] −1 −1 [6] −1 7 [7] −1 −1

`mylist.Insert(95);`
`mylist.Insert(54);`
`mylist.Delete(50);`
	The list:
	54
	58
	95
	54
		The list: starts in spot 1
		The first available spot is 0
		[0] 50 5
		[1] 54 2
		[2] 58 3
		[3] 95 4
		[4] 54 −1
		[5] −1 −1
		[6] −1 7
		[7] −1 −1

`mylist.Delete(95);`
`mylist.Delete(58);`
`mylist.Insert(25);`
	The list:
	54
	54
	25
		The list: starts in spot 1
		The first available spot is 3
		[0] 50 5
		[1] 54 4
		[2] 25 −1
		[3] 95 0
		[4] 54 2
		[5] −1 −1
		[6] −1 7
		[7] −1 −1

9.6 Write a Java function for an unordered dynamic LinkedList class to insert the node at the beginning of the list. The function heading would be **void rnsertBeginning(Item x);**

```
public void InsertBeginning(int x)  //inserts x and hooks to beginning of list
{
  if (head==null)                   //if list is empty, item is first
     head=new Node(x);
  else                              //insert in front
     head=new Node(x, head);
  size++;
}
```

9.7 What would be the output of the following consecutive sections of code using the Java ordered LinkedList class?

Consecutive Sections of Code	Output of paint()

`myList=new LinkedList();`
`int yloc=0;`
`for (int i=0; i<5; i++) myList.Insert((i+1)*3);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25,yloc+=15);`
	The list is
	3 6 9 12 15

`myList.Delete (9);`
`myList.Delete (15);`
`g.drawString ("The list is ",25, yloc+=15);`
`g.drawString (myList.toString(), 25, yloc+=15);`
	The list is
	3 6 12

`myList.Insert(2);`
`myList.Insert(34);`
`myList.Insert(23);`
`g.drawString ("The list is " 25, yloc+=15);`
`g.drawString (myList.toString(), 25, yloc+=15);`
	The list is
	2 3 6 12 23 34

9.8 One of the goals of object-oriented programming is to reuse as much code as possible. Which line(s) in Figs 9-7a and 9-7b would need to be changed to make the class a stack of integers instead of characters?

Answer: Only one line

typedef char Item; would change to typedef int Item;

All the rest of the code uses the generic Item rather than an explicit data type. Any other data type or class could be substituted for the char or int.

9.9 (a) Change the line in the C++ PrintStack() of Fig 9-7b function to print the stack all on one line.

(b) What would be the output of this C++ code section if "Hello to you!" is entered by the user at the prompt?

Note: (1) Assume the Anclude <string.h> is at the beginning of the program.

(2) cin.get(str,num)=the input of an entire string of up to num characters

(a) `cout«p->value«endl;` would change to `cout«p->value«" ,`

(b)

Consecutive Sections of Code	Output

`int i;`
`Stack myS;`
`char message[80];`
`cout<<"Enter one line of text"<<endl;`
`cin.getline(message, 80);`
`for (i=0; i<strlen(message); i++)`
`  myS.Push(message[i]);`
`myS.PrintStack();`
`cout<<endl;`
	Enter one line of text
	Hello to you!
	
	The STACK:
	! u o y o t o l l e H

`char ch=myS.Pop();`
`cout<<ch<<" was popped"<<endl;`
`myS.PrintStack();`
	! was popped
	The STACK:
	u o y o t o l l e H

9.10 What would be the output of this Java code section?

Consecutive Code Section	Output

`myq=new Queue();`
`int yloc=0;`
`String message="Hello to you!";`
`for (int i=0; i<message.length(); i++)`
`  myq.Enqueue(message.charAt(i));`
`g.drawString ("The queue is ",25, yloc+=15);`
`g.drawString (myq.toString(), 25,yloc+=15);`
	The queue is
	H e l l o t o y o u !

`char x=myq.Dequeue();`
`g.drawString`
`  (x+" was the character dequeued ",25, yloc+= 25);`
`g.drawString ("The queue is ",25, yloc+=25);`
`g.drawString (myq.toString(), 25, yloc+=15);`
	H was the character
	dequeued
	The queue is
	e l l o t o y o u !

9.11 Given the Java LinkedList class in Fig. 9-6, write a member function to add a given value to each element in the list.

Notes:

(1) For any traversal of the list, just go through the list from the beginning and get each value to add. Use the same algorithm of traversal anytime you want to go through the list to search for something or change each element.

(2) In the Node class, there is no function to set the value except the constructor. This problem requires a new function to be added to the Node class in addition to the function which is written for the LinkedList class.

Add to the Node class:

```
//sets the value of the  node
void setValue(int x)     (value=x;)
```

Add to the Linked List class

```
public void Add(int num)          //adds num to each item in the list
{
   Node current;
   current=head;
   while (current !=null)
   {
      current.setValue(current.getValue()+ num); //add num to each value
      current=current.getLink();
   }
}
```


HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP CÓ LỜI GIẢI

9.1 Loại ngôn ngữ nào được dùng trong sách này có các cấu trúc được tạo sẵn thuộc các kiểu sau đây

(a) chuỗi

(b) mảng

(c) lớp

(d) danh sách liên kết

9.2 Một vào ADT được định nghĩa trong sách này là: mảng, danh sách liên kết, hàng chồng và hàng đợi. Số nào trong số các định nghĩa này phù hợp cho các ứng dụng sau đây?

(a) Danh sách thực phẩm

(b) Mô phỏng sự kiểm tra một cửa hàng thực phẩm

(c) Chiến lược sắp xếp trong công ty ở đó những người đã làm việc trong một thời gian ngắn nhất được sắp xếp trước tiên

(d) Điểm số của sinh viên đại học

9.3 Bổ sung một hàm vào lớp LinkedList trong hình 9.4 để chèn hạng mục vào đầu danh sách. Tiêu đề của hàm này sẽ là:

```
void InsertBeginning(Item x);
```

Mã:

```
void LinkedList::InsertBeginning(Item x)
//Inserts an item x in first available spot at the begin-
   ning of the list
{
 int p, current;
 //get the first spot
if (avail==END) cout<<"\n LinkedList::Insert -- list is full \n";
else
{
   p=avail;
   avail=space[avail].link;      //next available spot
   space[p].value=x;             //fill value

   //hook it at the beginning
   space[p].link=head;
   head=p;

   size++;
  }//end else list is full
}
```


Bảng sau đây minh họa cách sử dụng hàm InsertBeginning (Item x)

Code	mylist.PrintList('o');	mylist.PrintList('a');
LinkedList mylist(5); Mylist.InsertBeginning(10); Mylist.InsertBeginning(14); Mylist.InsertBeginning(9); Mylist.InsertBeginning(3); Mylist.InsertBeginning(15);	The list: 15 3 9 14 10	The list: starts in spot 4 The first available spot is −1 [0] 10 −1 [1] 14 0 [2] 9 1 [3] 3 2 [4] 15 3 [5] −1 6 [6] −1 7 [7] −1 −1

9.4 Hãy vẽ một mảng trong hình 9.5 trước và sau khi xóa số 14.

Before	After deleting 14
[0] 10 1 head=0	[0] 10 1 head=0
[1] 12 2 avail=4	[1] 12 3 avail=2
[2] 14 3	[2] 14 4
[3] 15 −1	[3] 15 −1
[4] −1 −1	[4] −1 −1
[5] −1 6	[5] −1 6
[6] −1 7	[6] −1 7
[7] −1 −1	[7] −1 −1

9.5 Loại kết quả xuất nào trong số các mục các phần mã sau đây dành cho lớp C++ LinkedList tĩnh không theo thứ tự?

9.6 hãy viết một hàm Java dành cho lớp LinkedList động không theo thứ tự để chèn một nút ở đầu của danh sách. Tiêu đề hàm sẽ là void InsertBeginning (Item x);

9.7 Loại kết quả xuất nào trong các mục mã tuần tự sau đây sử dụng lớp Java ordered LinkedList?

9.8 Mục tiêu của lập trình hướng đối tượng đó là sử dụng lại càng nhiều mã càng tốt. Loại dòng nào trong hình 9.7a và 9.7b cần phải được thay đổi để tạo nên một lớp có một chồng các số nguyên thay vì các ký tự?

9.9 (a) Thay đổi dòng trong C++ PrintStack() của hàm số trong hình 9.7b để in ra chồng tất cả nội dung trên một dòng.

(b) Kết quả xuất của phần mã C++ sẽ là như thế nào nếu "Hello to you" được người dùng nhập vào tại dòng nhắc?

9.10 Kết quả xuất nào mà mục mã Java này đưa ra?

9.11 Cho lớp Java LinkedList như trong hình 9.6, hãy viết một hàm phần tử để bổ sung một giá trị đã cho vào một phần tử trong danh sách.

SUPPLEMENTARY PROBLEMS
Bài tập bổ sung

9.12 What is the difference between **static and dynamic** data structures?

9.13 Which type of ADT would be appropriate for the following applications?

(a) Simulation of car wash

(b) Packing and unpacking a suitcase

(c) Parking lot that has one lane and only one way in or out

(d) Checking incoming string to see if it is a palindrome

(e) Simulation of a bank line

(f) Todo list where all jobs have the same priority

(g) Employee records

9.14 Add a function to the LinkedList class class in Fig. 9-4 that would insert the item in order and create an ordered list. The function heading would be void **InsertInOrder(Item x);**

9.15 Draw the before and after array for Fig. 9-5 if the node 10 is deleted instead of the node 14.

9.16 What would be the output of the following sections of code for the unordered static C++ LinkedList class?

```
LinkedList mylist(7);
mylist.PrintList('o');
mylist.PrintList('a');
for (int i=2; i<6; i+=2)
   mylist.Insert(40-(3*i));
mylist.PrintList('o');
mylist.PrintList('a');
mylist.Insert(10);
mylist.Insert(20);
mylist.Insert(30);
mylist.Delete(34);
mylist.PrintList('o');
mylist.PrintList('a');

mylist.Delete(20);
mylist.Delete(30);
mylist.Insert(10);
```

```
mylist.PrintList('o');
mylist.PrintList('a');
```

9.17 Write a Java function for an unordered LinkedList class to insert the node at the end of the list. The function heading would be void ***InsertEnd(Item x);***

9.18 What would be the output of the following sections of code for the ordered dynamic Java LinkedList class?

```
myList=new LinkedList();
int yloc=0;
for (int i=2; i<7; i++) myList.Insert(i+5);
g.drawString ("The list is ",25, yloc+=15);
g.drawString (myList.toString(), 25,yloc+=15);

myList.Delete (9);
myList.Insert(2);
myList.Delete (11);
g.drawString ("The list is ",25, yloc+=15);
g.drawString (myList.toString(), 25, yloc+=15);

myList.Insert(34);
myList.Insert(7);
myList.Delete(8);
g.drawString ("The list is ",25, yloc+=15);
g.drawString (myList.toString(), 25, yloc+=15);
```

9.19 Using the PrintStack() from Solved Problem 9.9, what would be the output of this C++ code section?

```
int i;
Stack myS;
char word[30];
strcpy (word, "antidisestablishmentarianism");
for (i=0; i<strlen(word); i++)
    myS.Push(word[i]);
myS.PrintStack(); cout<<endl;

for (i=0; i<9; i++)
    myS.Pop();
myS.PrintStack(); cout<<endl;
```

9.20 What would be the output of this Java code section?

```
myq=new Queue();
int yloc=0;

String word="antidisestablishmentarianism";
for (int i=0; i<word.length(); i++)
   myq.Enqueue(word.charAt(i));
g.drawString ("The queue is ",25, yloc+=15);
g.drawString (myq.toString(), 25,yloc+=15);

for (int i=0; i<9; i++) myq.Dequeue();
g.drawString ("The queue is ",25, yloc+=25);
g.drawString (myq.toString(), 25, yloc+=15);
```

9.21 Given the Java LinkedList class in Fig. 9-6, write a member function which would find and return the number of times a given target item occurs in the list.

HƯỚNG DẪN ĐỌC HIỂU BÀI TẬP BỔ SUNG

9.12 *Có sự khác biệt nào giữa cấu trúc dữ liệu tĩnh và cấu trúc dữ liệu động?*

9.13 *Kiểu ADT nào phù hợp cho các trình ứng dụng sau đây?*

9.14 *Bổ sung một hàm vào lớp LinkedList class trong hình 9.4 để chèn một hạng mục theo thứ tự vào tạo một danh sách có thứ tự. Tiêu đề hàm sẽ là void InsertInOrder(Item x);*

9.15 *Hãy vẽ mảng trước và sau dùng cho hình 9.5 nếu nút số 10 được xóa bỏ thay vì nút số 14.*

9.16 *Kiểu kết quả xuất của mục mã sau đây dành cho lớp C++ LinkedList tĩnh theo thứ tự là gì?*

9.17 *Hãy viết một hàm Java dành cho một lớp LinkedList không theo thứ tự để chèn nút ở cuối danh sách. Tiêu đề của hàm sẽ là void InsertEnd(Item x);*

9.18 *Loại kết quả xuất của mục mã sau đây dành cho lớpJava LinkedList động được sắp theo thứ tự là gì?*

9.19 *Sử dụng PrintStack() từ bài tập 9.9, loại kết quả xuất của phần mã C++ này là gì?*

9.20 *Loại kết quả xuất củ a mục mã Java này là gì?*

9.21 *Cho lớp Java LinkedList trong hình 9.6, hãy viết một hàm phần tử qua đó tìm và trả về số lần mà một hạng mục đích đã cho xảy ra trong danh sách.*

ANSWERS TO SUPPLEMENTARY PROBLEMS
Giải đáp các bài tập bổ sung

9.12 **Static** data types use a fixed amount of memory even if not all locations are filled. Dynamic data types use pointer variables, and grow and shrink during the run of the program through the allocation and deallocation of memory.

9.13 ADTs:

(a) Queue
(b) Stack
(c) Stack
(d) Stack and queue together
(e) Queue
(f) Unordered linked list
(g) Ordered linked list

9.14 Code:


```
void LinkedList::InsertInOrder(Item x)
//Inserts an item x in first available spot and keep the list IN ORDER
{
    int p, prev, current;
    //get the first spot
    if (avail==END) cout<<"\n LinkedList::Insert -- list is full \n";
    else
    {
        p=avail;
        avail=space[avail].link; //next available spot
        space[p].value=x;        //fill values
        space[p].link=END;

        //find the correct spot to hook it up
        if (head==END)                  //hook it at the beginning
            head=p;
        else
        {
            current=head;
            prev=current;

            while (current!=END &&          //not the end
                space[current].value<x)     //in order
            {
```

```
            prev=current;
            current=space[current].link;
        }
        //now current points to item greater than x
        //OR points to first item in list
        if (current==prev)          //hook at beginning of list
        {
            space[p].link=head;
            head=p;
        }
        else                        //hook it up in proper sequence
        {
            space[p].link=space[prev].link;
            space[prev].link=p;
        }
    }//end else hook at end
    size++;
  }//end else list is full
}
```

9.15 Before and after arrays

Before				After Deleting 10			
[0]	10	1	head=0	[0]	10	4	head=1
[1]	12	2	avail=4	[1]	12	2	avail=0
[2]	14	3		[2]	14	3	
[3]	15	−1		[3]	15	−1	
[4]	−1	−1		[4]	−1	−1	
[5]	−1	6		[5]	−1	6	
[6]	−1	7		[6]	−1	7	
[7]	−1	−1		[7]	−1	−1	

9.16 Output:

Code	mylist.PrintList('o');	mylist.PrintList('a');
`LinkedList mylist(7);`	The list: Empty	The list: starts in spot −1 The first available spot is 0 [0] −1 1 [1] −1 2 [2] −1 3 [3] −1 4 [4] −1 5 [5] −1 6 [6] −1 −1 [7] −1 −1
`for (int i=2; i<6; i++)` `  mylist.Insert(40-(3*i));`	The list: 34 28	The list: starts in spot 0 The first available spot is 2 [0] 34 1 [1] 28 −1 [2] −1 3 [3] −1 4 [4] −1 5 [5] −1 6 [6] −1 −1 [7] −1 −1

`mylist.Insert(10);`
`mylist.Insert(20);`
`mylist.Insert(30);`
`mylist.Delete(34);`
	The list:
	28
	10
	20
	30
		The list: starts in spot 1
		The first available spot is 0
		[0] 34 5
		[1] 28 2
		[2] 10 3
		[3] 20 4
		[4] 30 −1
		[5] −1 6
		[6] −1 −1
		[7] −1 −1

`mylist.Delete(20);`
`mylist.Delete(30);`
`mylist.Insert(10);`
	The list:
	28
	10
	10
		The list: starts in spot 1
		The first available spot is 3
		[0] 34 5
		[1] 28 2
		[2] 10 4
		[3] 20 0
		[4] 10 −1
		[5] −1 6
		[6] −1 −1
		[7] −1 −1

9.17 Code:

```
public void InsertEnd(int x)      //inserts x and hooks to end of list
{
  Node current, p;
  if (head==null)                 //if list is empty, item is first
     head=new Node(x);
  else                            //find end and insert
  {
     current=head;
     while (current.getLink() !=null)
        current=current.getLink();
     p=new Node(x);               //sets link null
     current.setLink(p);          //hooks to end
  }
  size++;
}
```

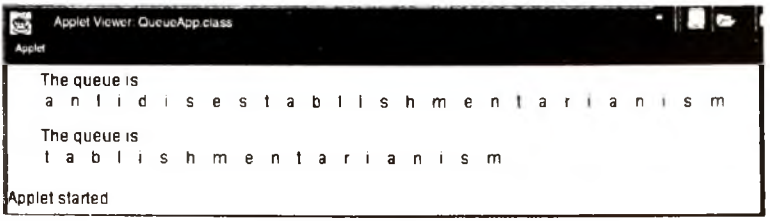
9.18 Output:

The list is
7891011
The list is
27810
The list is
277103.1

9.19 Output:

The STACK:
m s i n a i r a t n e m h s i l b a t s e s i d i t n a
The STACK:
n e m h s i l b a t s e s i d i t n a

9.20 Output:



9.21 Code:

```
public int Count(int target) //counts the number of times target is in the list
{
   int counter=0;
   Node current;
   current=head;
   while (current !=null)
   {
      if (current.getValue()==target) counter++; //add one to counter if there
      current=current.getLink();
   }
   return counter;
}
```

Contents

(MỤC LỤC)

Giáo trình tiếng Anh chuyên ngành

KHOA HỌC MÁY TÍNH

- Châu Văn Trung -

Chịu trách nhiệm xuất bản :
HOÀNG CHÍ DŨNG

Biên tập : Lê Tử Giang
Trình bày : Khắc Tùng
Vẽ bìa : Lê Thành

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

TỔNG PHÁT HÀNH

CÔNG TY VĂN HÓA NHÂN VĂN • 189 CMT8 - P.7 - Q. Tân Bình
TP. Hồ Chí Minh. ĐT: 9702389 - 9700420
NHÀ SÁCH NHÂN VĂN • 486 Nguyễn Thị Minh Khai - P.2 - Q.3
TP. Hồ Chí Minh. ĐT: 8396733
SIÊU THỊ SÁCH NHÂN VĂN • 394 Quốc lộ 15 - P. Trung Dũng
TP. Biên Hòa. ĐT: (061)917839
NHÀ SÁCH LÊ LAI • 171 Lê Lai - TP. Vũng Tàu. ĐT: (064) 5[illegible]359

In 1000 cuốn, khổ 16x24cm. Tại Xưởng in CN Trung Tâm Hội Chợ Triển Lãm Việt Nam.
Giấy đăng ký KHXB số: 151-2006/CXB/208-313/GTVT. In xong và nộp lưu chiểu tháng 5 năm 2006.