Table of Contents

Sessions

Session

1 Basics of C

Concepts

Objectives

At the end of this session, you will be able to:

- *Differentiate(phân biệt) between Command, Program and Software*
- *Explain(giải thích) the beginning of C*
- *Explain when and why is C used*
- *Discuss(thảo luận) the C program structure(cấu trúc chương trình)*
- *Discuss algorithms(thuật toán)*
- *Draw flowcharts(sơ đồ luồng công việc)*
- *List the symbo(kí hiệu)ls used in flowcharts*

Introduction

Today computers have pervaded every field. Automation is the key concept that is driving the world. Any kind of job requires some amount of knowledge of IT and programming. C is a high level programming language, which every programmer should know. Hence in this book, we will be studying the C language constructs in detail. To start with let us understand the difference between the words software, program and command.

1.1 Instructions to a Computer

When a computer is started, it automatically does some processes and comes to a particular screen. How does this happen? The answer is simple. The operating system software exists inside the computer. The operating system is referred to as system software. This software starts up the computer and performs some initial settings before giving us an operational screen. To achieve this, the operating system is made up of a set of programs. Every program tries to give solution to one or more problems. Each program is set of instruction to solve the problem it tries to address. Thus a group of instructions make up a program and a group of programs make up software.

Let's consider an analogy to get more clarity: One of our friends comes home and we serve the Strawberry Milk Shake. He finds it so tasty that he too wants the recipe. So, we give him the recipe as:

Concepts

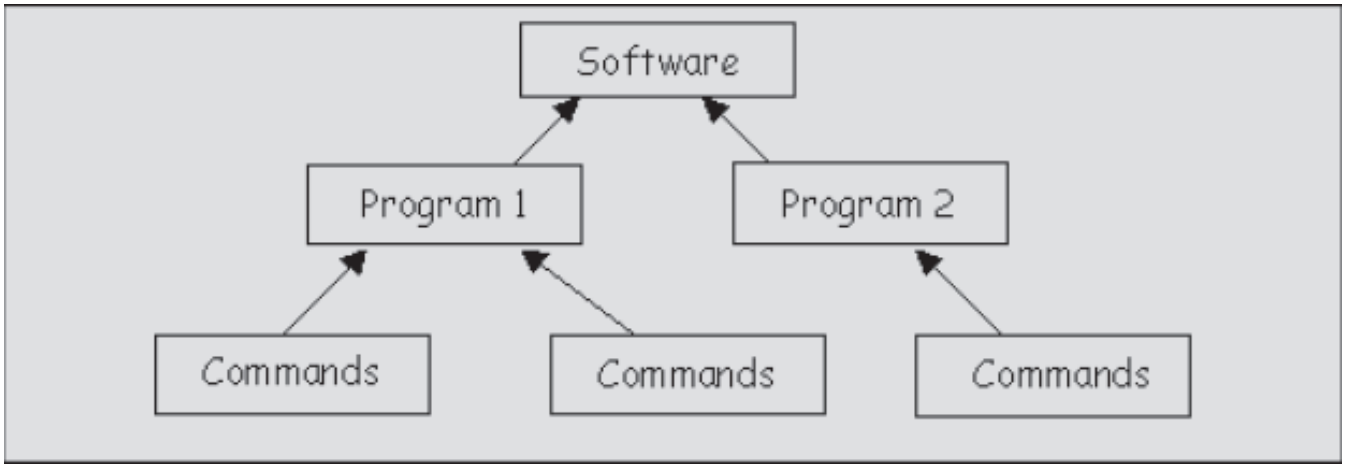


Figure 1.1: Software, Program and Commands

1.2 The C Language

While, BCPL and B do not support data-types (they are typeless), C provides a variety of data types. The main data- types are characters, integers and floating-point numbers.

C is closely associated with the UNIX system yet C is not tied to any one operating system or machine. C has been effectively used to write programs in different domains.

C was used for systems programming(C được sử dụng để lập trình hệ thống). A system program is associated with the operating system of the computer or its support utilities(Một chương trình hệ thống được liên kết với hệ điều hành của máy tính hoặc các tiện ích hỗ trợ của nó). . Operating Systems(hệ điều hành), Interpreters(trình thông dịch), Editors(trình biên tập), Assembly programs(chương trình lắp ráp) are usually called systems programs(thường được gọi là chương trình hệ thống). UNIX Operating System was developed using C(Hệ điều hành UNIX được phát triển bằng C). . C is now being used by many programmersfor kinds of tasksbecause of its portability and efficiency. C compilers are vailable for almost all computers. Codes written in C on a one machine can be compiled and run on another machine by making a few or no changes. C compiler produces fast and error-free object code.

C also offers the speed of an assembly language. Programmers can create and maintain library of functions, which can reused by other programs. Thus large projects can be managed easily, with minimum efforts.

1.2.1 C - A Middle Level Language

C is thought of as a middle-level language because it combines elements of high-level languages and functionalities of an assembly (low-level) language. C allows manipulation of the basic elements of a

computer i.e. bits, bytes, addresses etc. Also, C code is very portable(mã C rất dễ di chuyển), , that is, software written on one type of computer can work on another type of computer (nghĩa là phần mềm được viết trên một loại máy tính có thể hoạt động trên một loại máy tính khác). Although C has five basic built-in data types, it is not strongly typed language as compared to high-level languages. C allows data type conversions (C cho phép chuyển đổi kiểu dữ liệu). It allows direct manipulation of bits, bytes, words, and pointers (Nó cho phép thao tác trực tiếp các bit, byte, từ và con trỏ). Thus, it is used for system-level programming (nó được sử dụng cho lập trình cấp hệ thống)..

1.2.2 C - A Structured Language

C allows synthesis of code and data (C cho phép tổng hợp mã và dữ liệu.). This is a distinguishing feature of any structured language (Đây là một tính năng phân biệt của bất kỳ ngôn ngữ có cấu trúc nào). It refers to the ability of a language to collect and hide all information and instructions, necessary to perform a specific task, from the rest of the program (Nó đề cập đến khả năng của một ngôn ngữ trong việc thu thập và ẩn tất cả thông tin và hướng dẫn, cần thiết để thực hiện một nhiệm vụ cụ thể, khỏi phần còn lại của chương trình.). This can be done using functions or code blocks (Điều này có thể được thực hiện bằng cách sử dụng các hàm hoặc khối mã). Functions (hàm) are used to define and separate, tasks required in a program (được sử dụng để xác định và tách biệt các nhiệm vụ cần thiết trong một chương trình). Code block (Khối mã) is a logically connected group of program statements that is treated like a unit (là một nhóm các câu lệnh chương trình được kết nối logic được xử lý như một đơn vị). A code block is created by placing a sequence of statements between opening and closing curly braces as shown below (Một khối mã được tạo ra bằng cách đặt một chuỗi các câu lệnh giữa dấu ngoặc nhọn mở và đóng như minh họa bên dưới)

```
do
{
  i = i + 1;
  .
  .
  .
} while (i < 40);
```

Structured language support several loop constructs, such as while, do-while, and for. These loop constructs help the programmers to control the flow of the program.

1.3 The C Program Structure

C has few keywords, 32 to be precise (C có ít từ khóa, chính xác là 32). . These keywords, combined with the formal C syntax, form the C language. But many C compilers have added more keywords to use the memory organization of certain preprocessors.

Some rules(qui tắc) for programs written in C are as follows:

- All keywords are lower cased (Tất cả các từ khóa đều được viết thường)
- C is case sensitive, `do while` is different from **DO WHILE (C phân biệt chữ hoa chữ thường, do while khác với DO WHILE)**
- Keywords cannot be used for any other purpose, that is, they cannot be used as a variable or

Concepts

function name (Từ khóa không thể được sử dụng cho bất kỳ mục đích nào khác, nghĩa là chúng không thể được sử dụng như một biến hoặc tên hàm)

- `main()` is always the first function called when a program execution begins (main luôn là hàm đầu tiên được gọi khi chương trình bắt đầu thực thi)

Consider the following program code:

```
main ()
{
  /* This is a sample program */
  int i = 0;
  i = i + 1;
  .
  .
}
```

Note: Various aspects of a C program are discussed with respect to the above code. This code will be referred to as `sample _ code`, wherever applicable.

1.3.1 Function Definition

C programs are divided into units called functions. The `sample _ code` has only one function `main()`. The operating system always passes control to `main()` when a C program is executed (điều hành luôn chuyển quyền điều khiển cho hàm main() khi một chương trình C được thực thi).

The function name is always followed by parentheses .(Tên hàm luôn được theo sau bởi dấu ngoặc đơn). The parentheses may or may not contain parameters (Dấu ngoặc đơn có thể chứa hoặc không chứa tham số).

1.3.2 Delimiters

The function definition is followed by an pen curly brace ({) (Định nghĩa hàm được theo sau bởi dấu ngoặc nhọn mở ({)).This curly brace signals the beginning of the function. Similarly a closing curly brace (}) after the statements, in the function, indicate the end of the function. The opening brace ({) indicates that a code of block is about to begin and the closing brace (}) terminates the block of code. In `sample_code`, there are two statements between the braces. In addition to functions, the braces are also used to delimit blocks of code in other situations like loops and decision-making statements.

1.3.3 Statement Terminator

Consider the line `int i = 0` in `sample_code` is a statement. A statement in C is terminated with a semicolon (;) (Một câu lệnh trong C được kết thúc bằng dấu chấm phẩy (;)). . A carriage return, **whitespace**, or a **tab** is not understood by the C compiler (ký tự trả về, khoảng trắng hoặc tab : trình biên dịch C không hiểu được.

There can be more than one statement on the same line as long as each one of them is terminated with a semi-colon. A statement that does not end in a semicolon is treated as an invalid line of code in C.

1.3.4 Comment Lines

Comments are usually written to describe the task of a particular command, function or an entire program. The compiler ignores them. In C, comments begin with /* and are terminated with */(Trong C, chú thích bắt đầu bằng /* và kết thúc bằng */), in case the comments contain multiple lines(trong trường hợp chú thích chứa nhiều dòng.). Care should be taken that the terminating delimiter (*/) is not forgotten(Cần lưu ý không quên dấu phân cách kết thúc (*/)). Otherwise, the entire program will be treated like a comment(Nếu không, toàn bộ chương trình sẽ được coi như một bình luận). In case the comment contains just a single line you can use // to indicate that it is a comment(Trong trường hợp bình luận chỉ chứa một dòng duy nhất, bạn có thể sử dụng // để chỉ ra rằng đó là một bình luận.).

For example

```
int a=0; //Variable 'a' has been declared as an integer data type
```

1.3.5 The C Library

All C compilers come with a standard library of functions that perform the common tasks.(Tất cả các trình biên dịch C đều đi kèm với một thư viện tiêu chuẩn chứa các hàm thực hiện các tác vụ thông thường).In some installations of C, the library exists in one large file while in others it is contained in numerous small files.(Trong một số cài đặt C, thư viện tồn tại dưới dạng một tệp lớn, trong khi ở các cài đặt khác nó được chia thành nhiều tệp nhỏ.).While writing a program, the functions contained in the library can be used for various tasks.(Khi viết một chương trình, các hàm có trong thư viện có thể được sử dụng cho nhiều tác vụ khác nhau.)A function written by a programmer can be placed in the library and be used in as many programs as and when required.(Một hàm do lập trình viên viết có thể được đặt vào thư viện và sử dụng trong nhiều chương trình khác khi cần)Some compilers allow functions to be added in the standard library, while some compilers require a separate library to be created.(Một số trình biên dịch cho phép thêm các hàm vào thư viện tiêu chuẩn, trong khi một số khác yêu cầu tạo một thư viện riêng biệt)

1.4 Compiling and Running a Program

Thevariousstagesoftranslationofa Cprogramfromsourcecodetoexecutablecodeareasfollows:Các giai đoạn khác nhau trong quá trình dịch một chương trình C từ mã nguồn sang mã thực thi như sau:

- **Editor/Word Processor**

The source code is written using an editor or a word processor.(Mã nguồn được viết bằng một trình soạn thảo hoặc chương trình xử lý văn bản.)The code should be written in the form of standard text files, as C accepts source code only in this form (Mã phải được viết dưới dạng tệp văn bản tiêu chuẩn, vì C chỉ chấp nhận mã nguồn dưới dạng này).Some compilers supply programming environments (see appendix) that include an editor (Một số trình biên dịch cung cấp môi trường lập trình (xem phần phụ lục) bao gồm cả trình soạn thảo văn bản)

- **Source Code**

This is the text of the program, which the user can read.(Đây là văn bản của chương trình, mà người dùng có thể đọc được.)It is the input for the C compiler.(Đây là đầu vào cho trình biên dịch C)

- **C Preprocessor**

The source code is first passed through the C preprocessor.(Mã nguồn đầu tiên được chuyển qua bộ tiền xử lý C.)Preprocessors act on statements beginning with #.(Các bộ tiền xử lý hoạt động trên các câu lệnh bắt đầu bằng #).These statements are called directives (explained later).(Những câu lệnh này được gọi là chỉ thị)The directives are usually placed at the start of the program, though they can be placed anywhere else.(Các chỉ thị thường được đặt ở đầu chương trình, mặc dù chúng có thể được đặt ở bất kỳ đâu).The directives are short names given to a set of code.(Các chỉ thị là những tên ngắn được gán cho một tập hợp mã lệnh)

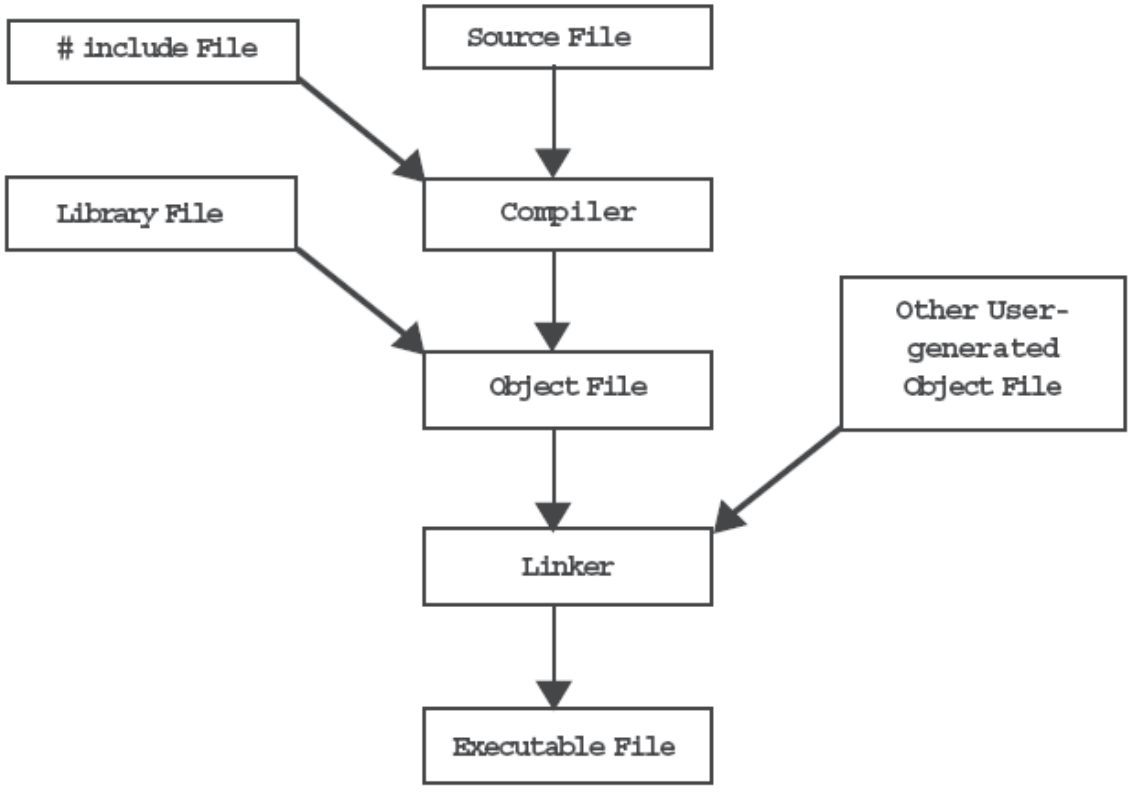


Figure 1.2: Compiling and Running a Program

1.5 The Programming Approach to Solving Problems

The steps would look as shown below:

STEP 1: Leave the room

STEP 2: Head towards the staircase

STEP 3: Go down to the basement

Concepts

STEP 4: Head for the cafeteria

An algorithm can be defined as a procedure, formula, or recipe for solving a problem.(Một thuật toán có thể được định nghĩa là một quy trình, công thức hoặc phương pháp để giải quyết một vấn đề.)It consists of a set of steps that help to arrive at a solution.(Nó bao gồm một tập hợp các bước giúp tìm ra giải pháp.)

From our discussion, it is obvious that in order to solve a problem, we need to first understand the problem.(Từ cuộc thảo luận của chúng ta, rõ ràng rằng để giải quyết một vấn đề, trước tiên chúng ta cần hiểu rõ vấn đề đó.)Next, we need to gather all relevant information required.(Tiếp theo, chúng ta cần thu thập tất cả thông tin liên quan cần thiết.)Once this is done, the next step would be to process these bits of information.(Sau khi hoàn thành, bước tiếp theo là xử lý những thông tin này.)Finally, we would arrive at the solution to the problem.(Cuối cùng, chúng ta sẽ tìm ra giải pháp cho vấn đề.)

The algorithm we have are a set of steps listed in simple language.(Thuật toán mà chúng ta có là một tập hợp các bước được liệt kê bằng ngôn ngữ đơn giản).It is very likely that though the steps written by two people may be similar, the language used to express these steps may be different.(Rất có khả năng rằng mặc dù các bước viết bởi hai người có thể tương tự, ngôn ngữ dùng để diễn đạt các bước này có thể khác nhau).It is, therefore, necessary to have some standard method of writing algorithms so that everyone easily understands it.(Do đó, cần phải có một phương pháp chuẩn hóa để viết thuật toán để mọi người đều có thể dễ dàng hiểu được).Hence, algorithms are written using two standard methods—pseudo codes and flowcharts (Vì vậy, thuật toán được viết bằng hai phương pháp chuẩn: mã giả và lưu đồ).

1.5.1 Pseudo code

Note that ‘pseudo code’ is not actual code (pseudo=false) (Lưu ý rằng 'mã giả' không phải là mã thực sự (pseudo = sai)).Pseudo code uses a certain standard set of words, which makes it resemble a code (Mã giả sử dụng một tập hợp từ chuẩn nhất định, khiến nó trông giống như mã lệnh).However, unlike code, pseudo code cannot be compiled or run (Tuy nhiên, không giống như mã lệnh, mã giả không thể được biên dịch hoặc chạy).
Let us, for example, consider the pseudo code written in Example 1 for displaying a ‘Hello World!’ message .(Ví dụ, hãy xem xét mã giả trong Ví dụ 1 để hiển thị thông báo 'Hello World!')

Example 1:

```
BEGIN
   DISPLAY ‘Hello World!’
END
```

As can be seen in the simple pseudo code above, each pseudo code must start with the word BEGIN or START, and end with END or STOP (Như có thể thấy trong mã giả đơn giản ở trên, mỗi mã giả phải bắt đầu bằng từ BEGIN hoặc START và kết thúc bằng END hoặc STOP).To display some value, the word DISPLAY or WRITE is used (Để hiển thị một giá trị nào đó, từ DISPLAY hoặc WRITE được sử dụng.).

Since the value to be displayed is a constant value in this case, the value (Hello World) is enclosed within quotes (Vì giá trị được hiển thị là một giá trị hằng trong trường hợp này, giá trị (Hello World) được đặt trong dấu ngoặc kép).Similarly, to accept a value from the user, the word INPUT or READ is used (Tương tự, để nhận một giá trị từ người dùng, từ INPUT hoặc READ được sử dụng).
To understand this better, let us have a look at the pseudo code (Refer to Example 2) for accepting two numbers from the user, and for displaying the sum of the two numbers (Để hiểu điều này rõ hơn, hãy cùng xem xét mã giả (Tham khảo Ví dụ 2) để nhận hai số từ người dùng và hiển thị tổng của hai số đó).

Example 2:

```
BEGIN
    INPUT A, B
    DISPLAY A + B
END
```

In this pseudo code, the user inputs two values, which are stored in memory and can be accessed as A and B respectively (Trong mã giả này, người dùng nhập hai giá trị, được lưu trữ trong bộ nhớ và có thể truy cập tương ứng là A và). Such named locations in memory are called variables (Các vị trí được đặt tên như vậy trong bộ nhớ được gọi là biến). The next step in the pseudo code displays the sum of the values present in variables A and B(Bước tiếp theo trong mã giả hiển thị tổng các giá trị có trong các biến A và B).
However, the same pseudo code can be modified to store the sum of the variables in a third variable and then display the value in this variable as shown in Example 3.

Tuy nhiên, cùng một mã giả có thể được sửa đổi để lưu trữ tổng các biến trong biến thứ ba và sau đó hiển thị giá trị trong biến này như trong Ví dụ

Example 3:

```
BEGIN
    INPUT A, B
    C = A + B
    DISPLAY C
END
```

A set of instructions or steps in a pseudo code is collectively called a construct (Một tập hợp các hướng dẫn hoặc các bước trong mã giả được gọi chung là một cấu trúc). There are three types of programming constructs - sequence, selection, and iteration constructs (Có ba loại cấu trúc lập trình - cấu trúc tuần tự, cấu trúc lựa chọn và cấu trúc lặp). In the pseudo codes, we have written above, we have used sequence constructs (Trong các mã giả mà chúng tôi đã viết ở trên, chúng tôi đã sử dụng các cấu trúc tuần tự). These are called so as they are instructions that are performed in a sequence, one after the other, starting from the top (Chúng được gọi như vậy vì chúng là các hướng dẫn được thực hiện theo trình tự, lần lượt, bắt đầu từ trên cùng.).

1.5.2 Flowcharts

To understand this better, let us have a look at a flowchart, given in figure 1.3, for displaying the traditional 'Hello World!' message.

Để hiểu rõ hơn về điều này, chúng ta hãy xem sơ đồ khối được đưa ra trong hình 1.3 để hiển thị phương pháp truyền thống
Tin nhắn 'Xin chào thế giới!'.

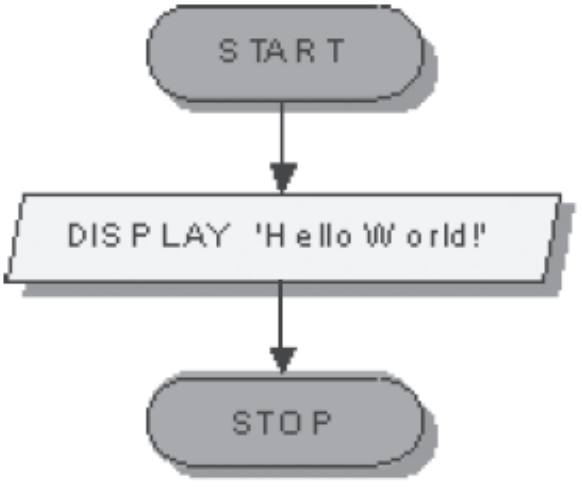


Figure 1.3: Flowchart to add two numbers

Flowcharts, like pseudo codes, begin with the START or BEGIN keyword, and end with the END or STOP keyword (Lưu đồ, giống như mã giả, bắt đầu với từ khóa START hoặc BEGIN và kết thúc với từ khóa END hoặc STOP).In a similar way, the DISPLAY keyword is used to display some value to the user (Theo cách tương tự, từ khóa DISPLAY được sử dụng để hiển thị một giá trị nào đó cho người dùng).However, here, every keyword is enclosed within symbols (Tuy nhiên, ở đây, mỗi từ khóa được đặt trong các ký hiệu).The different symbols used in flowcharting and their relevance are tabulated in Figure 1.4 (Các ký hiệu khác nhau được sử dụng trong việc vẽ lưu đồ và ý nghĩa của chúng được trình bày trong Hình 1.4).

Symbol	Description
	Start or End of the Program
	Computational Steps
	Input / Output instructions
	Decision making & Branching
	Connectors
	Flow Line

Fig 1.4: Flowchart Symbols

Example: We accepted two numbers from the user, added them up (stored the result in a third variable), and displayed the result (Ví dụ: Chúng ta đã nhận hai số từ người dùng, cộng chúng lại (lưu kết quả vào một biến thứ ba) và hiển thị kết quả).The flowchart for this would look as shown in Figure 1.5 (Lưu đồ cho điều này sẽ trông giống như được hiển thị trong Hình 1.5).

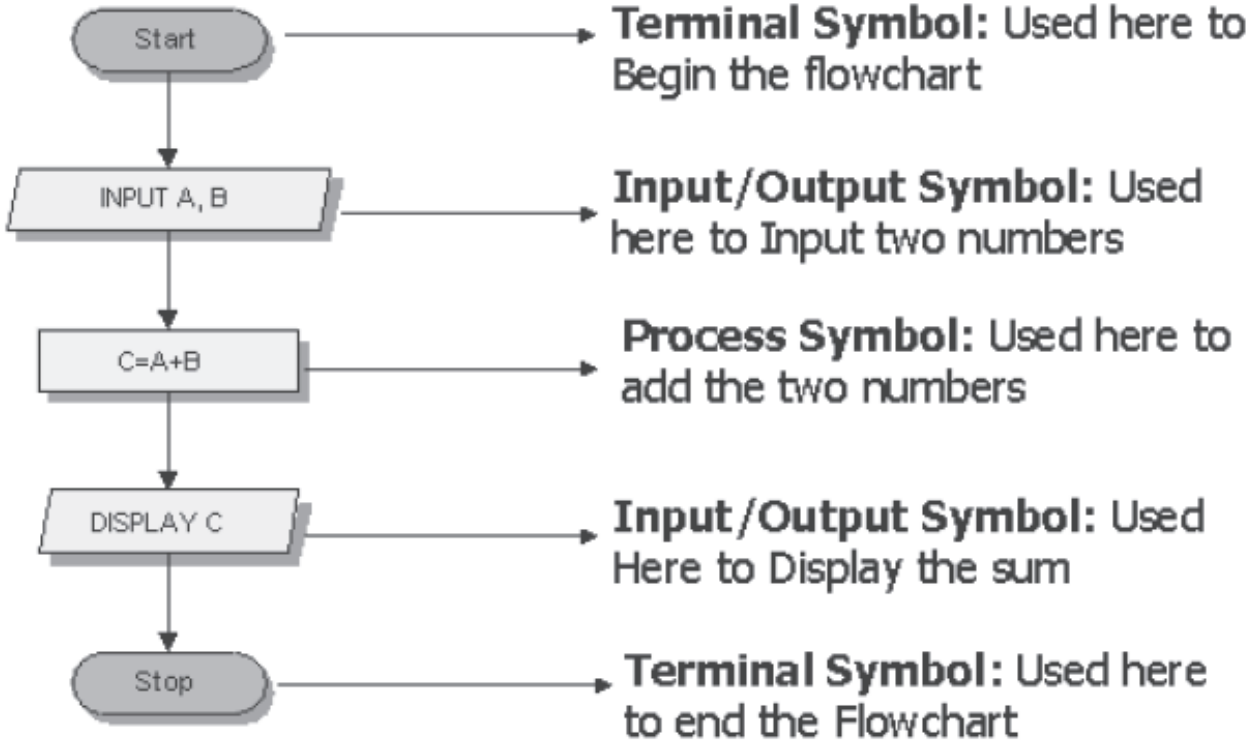


Fig 1.5: Flowchart to add to add two numbers

The step in which the values of the two variables are added and assigned to a third variable is considered to be a process and is shown by a rectangle (Bước trong đó giá trị của hai biến được cộng lại và gán cho một biến thứ ba được coi là một quá trình và được biểu thị bằng hình chữ nhật).

Most often, flowcharts scan several pages (Thường thì, lưu đồ sẽ trải dài qua nhiều trang).In such a case, symbols called connectors are used to indicate the location of the joins (Trong trường hợp như vậy, các ký hiệu được gọi là các kết nối được sử dụng để chỉ ra vị trí của các điểm nối).They help us identify the flow across the pages (Chúng giúp chúng ta xác định luồng qua các trang).Circles represent connectors and need to contain a letter or a number, as shown in Figure 1.6 (Các hình tròn đại diện cho các kết nối và cần chứa một chữ cái hoặc một số, như được hiển thị trong Hình 1.6).Using these, we can pick up the link between two incomplete flowcharts (Bằng cách sử dụng những điều này, chúng ta có thể lấy được liên kết giữa hai lưu đồ chưa hoàn chỉnh).

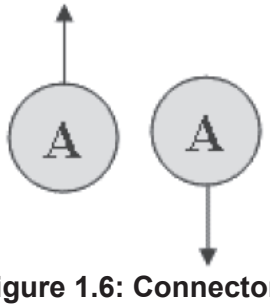


Figure 1.6: Connectors

Concepts

Some other essential things to be taken care of when drawing a flowchart are:(Một số điều cần thiết khác cần lưu ý khi vẽ sơ đồ luồng công việc là):

- Initially concentrate only on the logic of the problem and draw out the main path of the flowchart (Ban đầu chỉ tập trung vào logic của vấn đề và vạch ra lộ trình chính của sơ đồ.)
- A flowchart must have only one START and one STOP point (Sơ đồ luồng công việc chỉ được có một điểm BẮT ĐẦU và một điểm DỪNG.)
- It is not necessary to represent each and every step of a program in the flowchart. Only the essential and meaningful steps need to be represented (Không cần thiết phải thể hiện từng bước của một chương trình trong sơ đồ luồng. Chỉ có các bước thiết yếu và có ý nghĩa cần phải được trình bày).

We have worked with sequence constructs in which execution flows through all the instructions from the top in a sequence (Chúng ta đã làm việc với các cấu trúc tuần tự, trong đó quá trình thực thi chảy qua tất cả các lệnh từ trên xuống dưới theo một trình tự) .We can come across conditions in our program, based on which the path of execution may branch (Chúng ta có thể gặp các điều kiện trong chương trình của mình, dựa trên đó mà đường đi của quá trình thực thi có thể phân nhánh) .Such constructs are referred to as selection, conditional, or branching construct (.Các cấu trúc như vậy được gọi là cấu trúc chọn lựa, điều kiện hoặc phân nhánh).

- **The IF construct**

A basic selection construct is an 'IF' construct (Một cấu trúc chọn lựa cơ bản là cấu trúc 'IF) '.To understand this construct, let us consider an example where the customer is given a discount if he makes a purchase of above $100 (Để hiểu rõ cấu trúc này, hãy xem xét một ví dụ trong đó khách hàng được giảm giá nếu họ mua hàng trên 100 đô la) .Every time a customer is billed, a part of the code has to check if the bill amount exceeds $100 (Mỗi khi một khách hàng được thanh toán, một phần của mã cần kiểm tra xem số tiền trên hóa đơn có vượt quá 100 đô la hay không).If it does, then deduct 10% of the total amount; otherwise, deduct nothing (Nếu có, thì trừ 10% tổng số tiền; nếu không, thì không trừ gì cả)

The general form of an IF statement or construct is as follows:

```
IF condition
    Statements → Body of the IF construct
END IF
```

An 'IF' construct begins with IF followed by the condition (Một cấu trúc 'IF' bắt đầu bằng từ IF theo sau là điều kiện) .If the condition evaluates to true, then control is passed to the statements within its bod (Nếu điều kiện đánh giá là đúng, thì quyền điều khiển sẽ được chuyển đến các câu lệnh trong thân của nó) .If the condition returns false, the statements within the body are not executed, and the program flow continues with the next statement after the END IF (Nếu điều kiện trả về sai, các câu lệnh trong thân sẽ không được thực thi, và luồng chương trình sẽ tiếp tục với câu lệnh tiếp theo sau END IF) .The IF construct must always end with an END IF, as it marks the end of the construct (Cấu trúc IF phải luôn kết thúc bằng END IF, vì nó đánh dấu sự kết thúc của cấu trúc).

Let us take a look at Example 4, which uses IF (Hãy cùng xem xét Ví dụ 4, trong đó sử dụng IF).

To find if a number is even, we proceed as follows (Để tìm xem một số có phải là số chẵn hay không, chúng ta tiến hành như sau):

Example 4:

```
BEGIN
    INPUT num
    r = num MOD 2
    IF r=0
      Display "Number is even"
    END IF
END
```

The above code accepts a number from the user, performs the MOD operation on it, and checks if the remainder is zero (Mã ở trên nhận một số từ người dùng, thực hiện phép MOD trên nó và kiểm tra xem phần dư có bằng không hay không) .If it is, then it displays a message; otherwise, it just exits. (Nếu có, thì nó hiển thị một thông báo; nếu không, nó chỉ thoát)

A flowchart for the above pseudo code would look as shown in Figure 1.7 (Lưu đồ cho mã giả ở trên sẽ trông giống như được hiển thị trong Hình 1.7).

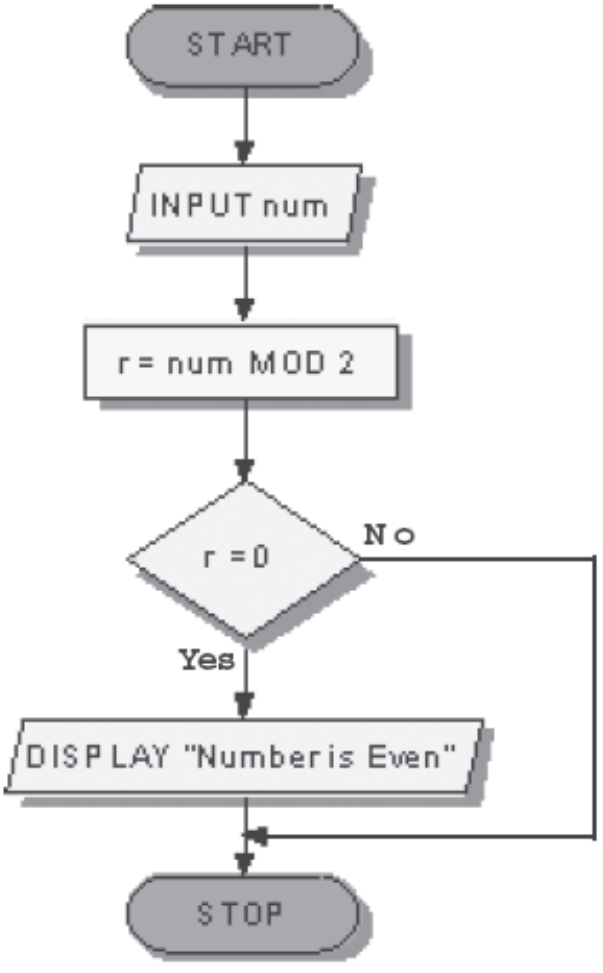


Figure 1.7 : Even Number or Not

The syntax for the IF statement in C is as follows:

```
if (condition)
{
  statements
}
```

- **The IF…ELSE Construct**

In Example 4, it will be nicer if we responded with a message saying that the number is not even (therefore odd) instead of just exiting (Trong Ví dụ 4, sẽ tốt hơn nếu chúng ta phản hồi bằng một thông điệp nói rằng số đó không phải là số chẵn (do đó là số lẻ) thay vì chỉ thoát) .One way of achieving this can be by adding another IF statement which checks if the number is not divisible (Một cách để đạt được điều này là thêm một câu lệnh IF khác kiểm tra xem số đó có phải là số không chia hết hay không).Refer to Example 5:

Example 5:

```
BEGIN
    INPUT num
    r = num MOD 2
    IF r=0
      DISPLAY "Even number"
    END IF
    IF r<>0
      DISPLAY "Odd number"
    END IF
END
```

Programming languages provide us with what is known as an IF…ELSE construct (Các ngôn ngữ lập trình cung cấp cho chúng ta một cấu trúc được gọi là IF…ELSE) .Using this would be a better and more efficient approach to solving the above problem (Việc sử dụng cấu trúc này sẽ là một cách tiếp cận tốt hơn và hiệu quả hơn để giải quyết vấn đề ở trên).The IF…ELSE construct enables the programmer to make a single comparison and then execute the steps depending on whether the result of the comparison is True or False (Cấu trúc IF…ELSE cho phép lập trình viên thực hiện một so sánh duy nhất và sau đó thực hiện các bước tùy thuộc vào việc kết quả của phép so sánh là Đúng hay Sai) .

The general form of the IF…ELSE statement is as follows:

```
IF condition
  Statement set1
ELSE
  Statement set2
END IF
```

The syntax for the `if…else` construct in C is given below.

```
if(condition)
{
  statement set1
}
else
{
  statement set2
}
```

If the condition evaluates to True, statement set1 is executed; otherwise, statement set2 is executed, but never both (Nếu điều kiện đánh giá là Đúng, thì tập hợp câu lệnh set1 được thực thi; nếu không, tập hợp câu lệnh set2 sẽ được thực thi, nhưng không bao giờ cả hai).So, a more efficient code for our even number example (Example 5) would be as shown in Example 6 (Vì vậy, mã hiệu quả hơn cho ví dụ về số chẵn của chúng ta (Ví dụ 5) sẽ được hiển thị như trong Ví dụ 6).

Example 6:

```
BEGIN
   INPUT num
   r=num MOD 2
   IF r=0
     DISPLAY "Even Number"
   ELSE
     DISPLAY "Odd Number"
   END IF
END
```

The flowchart for the above pseudo code is displayed in Figure 1.8.

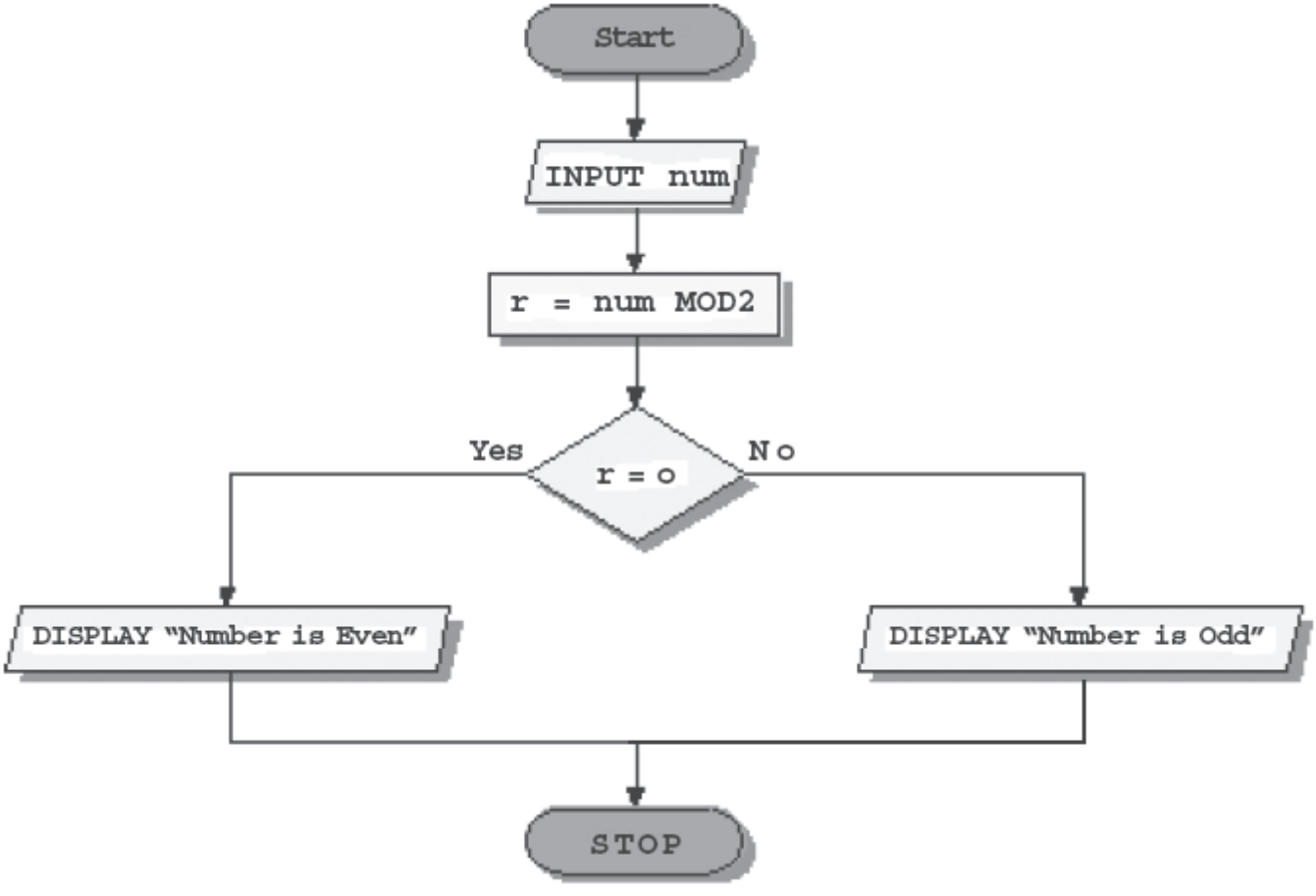


Figure 1.8: Even or Odd number

➢ **Multiple criteria using AND/OR**

The IF…ELSE construct helps to reduce complexity and enhances efficiency (Cấu trúc IF…ELSE giúp giảm độ phức tạp và tăng cường hiệu suất) .To some extent, it also improves the readability of the code 9Ở một mức độ nào đó, nó cũng cải thiện khả năng đọc của mã) .The IF examples we have discussed till now were fairly simple (Các ví dụ về IF mà chúng ta đã thảo luận cho đến bây giờ khá đơn giản).They had one IF condition to evaluate 9Chúng chỉ có một điều kiện IF để đánh giá) .Sometimes we have to check for more than one condition (Đôi khi chúng ta phải kiểm tra nhiều hơn một điều kiện)
for example.; ví dụ.This is where the AND operator can be used with the 'IF' statement (Đây là lúc toán tử AND có thể được sử dụng trong câu lệnh 'IF).Refer to Example 7

Example 7:

```
BEGIN
    INPUT yearsWithUs
    INPUT bizDone
    IF yearsWithUs >= 10 AND bizDone >= 5000000
      DISPLAY "Classified as an MVS"
    ELSE
      DISPLAY "A little more effort required!"
    END IF
END
```

Even there, we can conveniently use the AND operator to connect them just like how we used it (Tại đây, chúng ta thay thế toán tử AND bằng toán tử OR) .Remember we had learned in our previous section that the OR operator evaluates to True even if one of the conditions is True (Hãy nhớ rằng chúng ta đã học trong phần trước rằng toán tử OR đánh giá là Đúng ngay cả khi một trong các điều kiện là Đúng).

➢ **Nested IFs**

Another way to do Example 7 involves using Nested IFs (Một cách khác để thực hiện Ví dụ 7 là sử dụng các câu lệnh IF lồng nha) u.A nested IF is an IF inside another IF statement (Một câu lệnh IF lồng nhau là một câu lệnh IF nằm bên trong một câu lệnh IF khác) .Let us re-write the pseudo code in Example 7 using nested IFs (Hãy cùng viết lại mã giả trong Ví dụ 7 bằng cách sử dụng các câu lệnh IF lồng nhau) .The code would look like in Example 8 (Mã sẽ trông giống như trong Ví dụ 8).

Example 8:

```
BEGIN
    INPUT yearsWithUs
    INPUT bizDone
```

```
    IF yearsWithUs >= 10
      IF bizDone >= 5000000
        DISPLAY "Classified as an MVS"
      ELSE
        DISPLAY "A little more effort required!"
      END IF
    ELSE
      DISPLAY "A little more effort required!"
    END IF
END
```

The above code performs the same function without an 'AND'.

The flowchart for the pseudo code in example 8 is displayed in Figure 1.9.

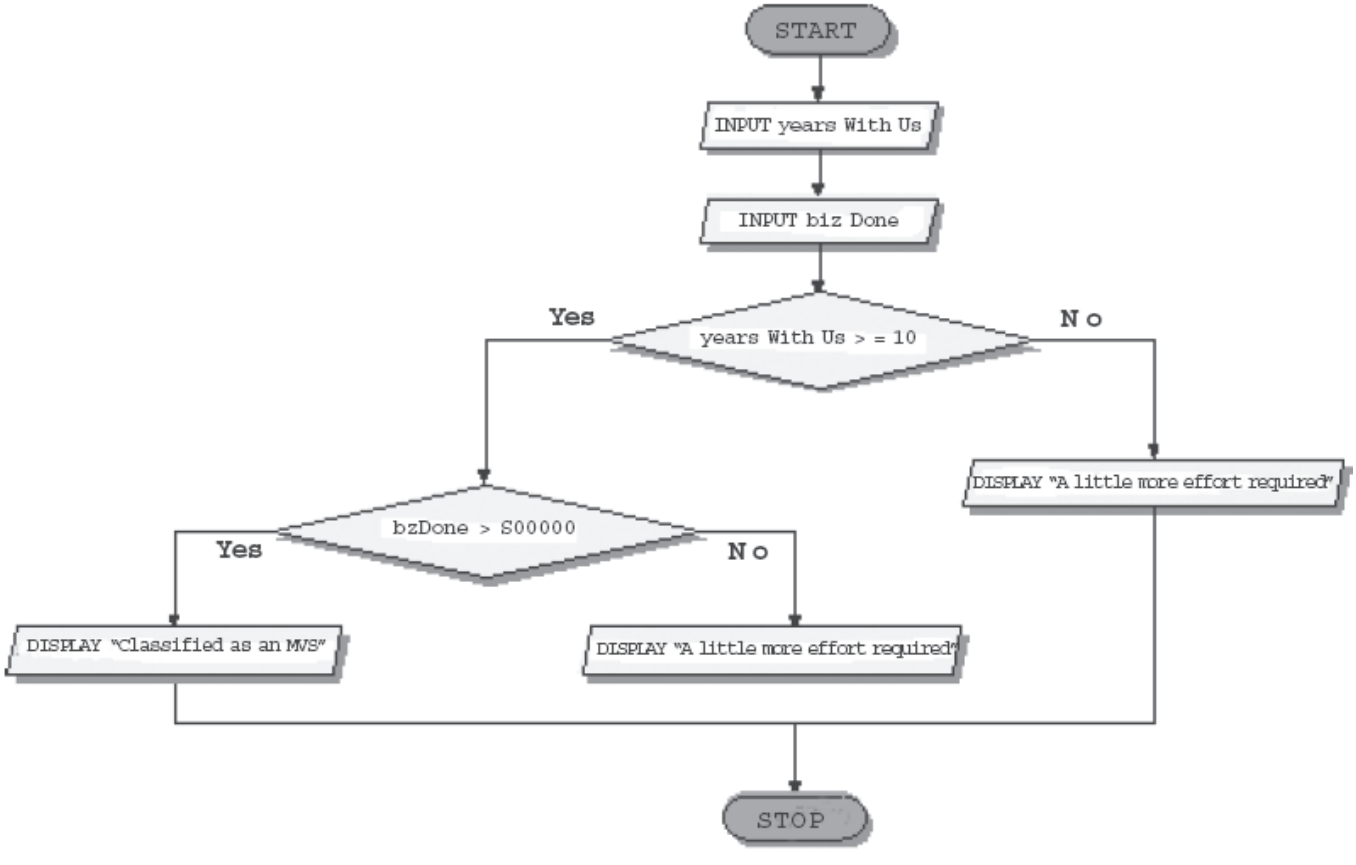


Figure 1.9: Nested IFs

A company allots basic salary to its employees based on the criteria specified in Table 1.1.

Grade	Experience	Salary
E	2	2000
E	3	3000
M	2	3000
M	3	4000

Table 1.1: Basic Salary

Therefore, for example, an employee with grade E and two years of experience would receive a salary of $2000, and an employee with the same grade but 3 years of experience would get a salary of $3000 (Do đó, ví dụ, một nhân viên có cấp bậc E và hai năm kinh nghiệm sẽ nhận được mức lương 2000 đô la, và một nhân viên có cùng cấp bậc nhưng có 3 năm kinh nghiệm sẽ nhận được mức lương 3000 đô la)

The pseudo code for the problem using the AND operator would be as in Example 9 (Mã giả cho vấn đề này sử dụng toán tử AND sẽ được trình bày như trong Ví dụ 9) .

Example 9:

```
BEGIN
    INPUT grade
    INPUT exp
    IF grade = ”E” AND exp = 2
      salary=2000
    ELSE
      IF grade = “E” AND exp= 3
        salary=3000
      END IF
    END IF
    IF grade = “M” AND exp = 2
      salary=3000
    ELSE
      IF grade = “M” AND exp = 3
        salary=4000
     END IF
  END IF
END
```

Now let us consider the modified pseudo code using nested IFs in Example 10.

Example 10:

```
BEGIN
    INPUT grade
    INPUT exp
    IF grade=”E”
      IF exp=2
        salary=2000
      ELSE
        IF exp=3
          salary=3000
        END IF
      END IF
    ELSE
     IF grade=”M”
       IF exp=2
         Salary=3000
       ELSE
        IF exp=3
          Salary=4000
        END IF
       END IF
     END IF
    END IF
```

At first look, the above code may look a little awry.Thoạt nhìn, mã ở trên có thể trông hơi lộn xộn.However, it is much more efficient.Tuy nhiên, nó hiệu quả hơn rất nhiều.Let us take the same example.Hãy cùng xem xét ví dụ tương tự.If an employee has a grade of E and has two years of experience, then his salary is calculated to be $2000 right in the first step of the first IF.Nếu một nhân viên có cấp bậc E và có hai năm kinh nghiệm, thì mức lương của họ được tính là 2000 đô la ngay trong bước đầu tiên của câu lệnh IF đầu tiên.After this, the code is exited as it does not need to execute any of the ELSE clauses.Sau đó, mã sẽ thoát ra vì không cần thực thi bất kỳ điều khoản ELSE nào.It therefore does not go through unnecessary IFs.Vì vậy, nó không đi qua các câu lệnh IF không cần thiết.So the code is more efficient, and the program runs faster.Vì vậy, mã này hiệu quả hơn và chương trình chạy nhanh hơn.

- **Loops**

A computer program, is a set A computer program is a set of statements, which are normally executed one after the other.Một chương trình máy tính là một tập hợp các câu lệnh, thường được thực thi lần lượt.It may be required to repeat certain steps a specific number of times or till a certain condition is met.Có thể cần lặp lại một số bước nhất định một số lần cụ thể hoặc cho đến khi một điều kiện nhất định được đáp ứng.

It would be easier if we could write the DISPLAY statement once and ask the computer to execute it 1000 times.Sẽ dễ hơn nếu chúng ta có thể viết câu lệnh DISPLAY một lần và yêu cầu máy tính thực thi nó 1000 lần.In order to do this, we need to include the DISPLAY statement in a loop construct and instruct the computer to loop through the statement 1000 times.Để làm điều này, chúng ta cần đưa câu lệnh DISPLAY vào một cấu trúc vòng lặp và hướng dẫn máy tính lặp lại câu lệnh đó 1000 lần.A rough pseudo code of what a looping construct would look like is given in Example 12.Một mã giả thô về cách mà một cấu trúc vòng lặp sẽ trông như thế nào được đưa ra trong Ví dụ 12.

Example 12:

```
Do loop 1000 times
  DISPLAY "Scooby"
end loop
```

The block of statement(s) [in this case just the DISPLAY statement] that appear in between the do loop and end loop statements get executed 1000 times.Khối câu lệnh (trong trường hợp này chỉ là câu lệnh DISPLAY) nằm giữa các câu lệnh do loop và end loop sẽ được thực thi 1000 lần.

These statements, along with the do loop and end loop statements, are called looping or iterative constructs.Các câu lệnh này, cùng với các câu lệnh do loop và end loop, được gọi là cấu trúc lặp hoặc cấu trúc lặp lại.Loops enable programmers to develop concise programs which otherwise would require thousands of statements to perform.Các vòng lặp cho phép lập trình viên phát triển các chương trình ngắn gọn, nếu không sẽ yêu cầu hàng nghìn câu lệnh để thực hiện.

Here we have used Do loop…end loop to indicate a general form of a loop.Ở đây, chúng ta đã sử dụng Do loop…end loop để chỉ ra một hình thức chung của vòng lặp.Another way of writing the same code is shown in Example 13.Một cách khác để viết cùng mã đó được hiển thị trong Ví dụ 13.

Example 13:

```
BEGIN
   cnt=0
   WHILE (cnt < 1000)
   DO
     DISPLAY "Scooby"
     cnt=cnt+1
   END DO
END
```

The flowchart for the pseudo code in Example 13 would look as shown in Figure 1.10.

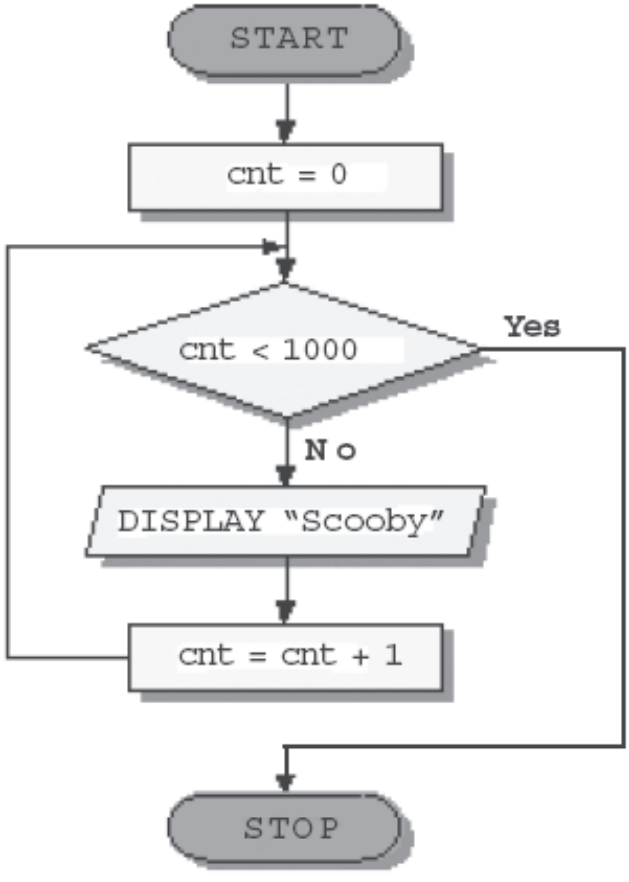


Figure 1.10: Loops

Notice that in Figure 1.10 there is no special symbol for showing loops. We use the branching symbol to check the condition and manage the flow of the program using flow-lines.

Concepts

Summary

Software is a set of programs.Phần mềm là một tập hợp các chương trình.
A program is a set of instructions.Một chương trình là một tập hợp các hướng dẫn.
Code blocks form the base of any C program.Các khối mã tạo thành nền tảng của bất kỳ chương trình C nào.

The C language has 32 keywords.Ngôn ngữ C có 32 từ khóa.
Steps involved in solving a problem are studying the problem in detail, gathering the relevant information, processing the information, and arriving at the results.Các bước liên quan đến việc giải quyết một vấn đề bao gồm nghiên cứu chi tiết vấn đề, thu thập thông tin liên quan, xử lý thông tin và đi đến kết quả.

An algorithm is a logical and concise list of steps to solve a problem.Thuật toán là một danh sách các bước hợp lý và ngắn gọn để giải quyết một vấn đề.
Algorithms are written using pseudo codes or flowcharts.Các thuật toán được viết bằng mã giả hoặc biểu đồ dòng chảy.

A pseudo code is a representation of an algorithm in a language that resembles code.Mã giả là một biểu diễn của một thuật toán trong một ngôn ngữ giống như mã.
A flowchart is a diagrammatic representation of an algorithm.Biểu đồ dòng chảy là một biểu diễn hình ảnh của một thuật toán.

The basic selection construct is an ‘IF’ construct.Cấu trúc chọn cơ bản là một cấu trúc 'IF'.
Flowcharts can be broken into parts, and connectors can be used to indicate the location of the joins.Biểu đồ dòng chảy có thể được chia thành các phần, và các đầu nối có thể được sử dụng để chỉ ra vị trí của các điểm nối.

When we come across a condition based on which the path of execution may branch, such constructs are referred to as selection, conditional, or branching constructs.Khi chúng ta gặp một điều kiện mà theo đó đường đi của việc thực hiện có thể phân nhánh, các cấu trúc như vậy được gọi là cấu trúc lựa chọn, điều kiện hoặc phân nhánh.

The IF…ELSE construct enables the programmer to make a single comparison and then execute the steps depending on whether the result of the comparison is True or False.Cấu trúc IF…ELSE cho phép lập trình viên thực hiện một phép so sánh đơn lẻ và sau đó thực hiện các bước tùy thuộc vào việc kết quả của phép so sánh là Đúng hay Sai.

A nested IF is an IF inside another IF statement.Một IF lồng nhau là một IF nằm trong một câu lệnh IF khác.

Often, it is necessary to repeat certain steps a specific number of times or till some specified condition is met.Thường thì cần lặp lại một số bước nhất định một số lần cụ thể hoặc cho đến khi một điều kiện nhất định được đáp ứng.The constructs which achieve these are known as iterative or looping constructs.Các cấu trúc đạt được điều này được gọi là các cấu trúc lặp hoặc vòng lặp.

Check Your Progress

1. C allows _____ of code and data.

2. A _____ is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive at a solution.

3. Flowcharts help us review and debug programs easily. (True / False)

4. A flowchart can have any number of start and stop points. (True / False)

5. A ____ is basically the execution of a sequence of statements until a particular condition is True or False.

Try It Yourself

1. Write a pseudo code and draw a flowchart to accept a value in degrees Celsius and to convert it into Fahrenheit. [Hint: C/5 = (F-32)/9]

2. Write a pseudo code and flowchart to accept a student's marks in Physics, Chemistry, and Biology. The total of these marks as well as the average should be displayed.

Session

2 Variables and Data Types

Concepts

Objectives

At the end of this session, you will be able to:

- *Discuss variables*
- *Differentiate between variables and constants*
- *List the different data types and make use of them in C programs*
- *Discuss arithmetic operators*

Introduction

2.1 Variables

Earlier, programmers had to write their programs in the language of the machine, tức là, bằng ngôn ngữ của máy tính, that is, in 1's and 0's tức là, bằng 1 và 0 . If the programmer wanted to temporarily store a value, nếu lập trình viên muốn lưu trữ tạm thời một giá trị, the exact location where the data is stored inside the memory of the computer had to be assigned. Vị trí chính xác nơi dữ liệu được lưu trữ trong bộ nhớ của máy tính phải được chỉ định. This storage location had to be a specific number và vị trí lưu trữ này phải là một số cụ thể or memory address. hoặc địa chỉ bộ nhớ.

Modern day languages enable us to use symbolic names known as variables, các ngôn ngữ hiện đại cho phép chúng ta sử dụng các tên ký hiệu được gọi là biến. to refer to the memory location where a particular value is to be stored. để chỉ đến vị trí bộ nhớ nơi một giá trị cụ thể sẽ được lưu trữ.

The data type decides the amount of memory to be assigned. Kiểu dữ liệu quyết định lượng bộ nhớ cần được chỉ định. The names that we assign to variables help us in reusing the data as and when required. Các tên mà chúng ta gán cho biến giúp chúng ta tái sử dụng dữ liệu khi cần thiết.

Session 2

Variables and Data Types

Concepts

We are familiar with the use of letters, which represent quantities in a formula. For example, the area of a rectangle is given by:

```
Area = A = Length x Breadth = L x B
```

The simple interest is given by:

```
Interest = I = Principal x Time x Rate / 100 = P x T x R / 100
```

The letters **A**, **L**, **B**, **I**, **P**, **T**, **R**, are all variables and are short notations that represent various values.

Consider the following example:

The sum of the marks obtained by 5 students is to be displayed. The sum can be displayed with the help of the following instruction.

```
Display the sum of 24, 56, 72, 36 and 82.
```

Once the sum is displayed, it is lost from the computer's memory. Suppose we want to calculate the average of the marks, the sum would have to be recalculated.

A better approach would be to store the result in the computer's memory, and reuse it as and when required.

```
sum = 24 + 56+ 72 + 36 + 82
```

Here, **sum** is a variable that is used to store the sum of the five numbers. When we have to calculate average, it can be done as follows:

```
Avg = sum / 5
```

In C, all variables should be declared before they can be used. Let us, for example, take the example of accepting two numbers and displaying their sum. The code for the same is provided in Example 1.

Example 1:

```
BEGIN
    DISPLAY 'Enter 2 numbers'
    INPUT A, B
    C = A + B
    DISPLAY C
END
```

A, B, and C in the code above are variables. A, B và C trong đoạn mã trên là các biến. The variable names save you from remembering memory locations by address. Các tên biến giúp bạn không cần nhớ các địa chỉ bộ nhớ. When code is written and executed, the operating system takes care of allocating some memory space for these variables. Khi mã được viết và thực thi, hệ điều hành sẽ lo liệu việc phân bổ một số không gian bộ nhớ cho các biến này. The operating system maps a variable name to a specific memory location. Hệ điều hành ánh xạ một tên biến đến một vị trí bộ nhớ cụ thể. Now, to refer to a particular value in memory, we need to specify just the variable name. Bây giờ, để chỉ đến một giá trị cụ thể trong bộ nhớ, chúng ta chỉ cần chỉ định tên biến.In the example above, two values are accepted from the user, and these are stored in some memory location. Trong ví dụ trên, hai giá trị được chấp nhận từ người dùng và được lưu trữ trong một số vị trí bộ nhớ. These memory locations can be accessed via the variable names A and B. Các vị trí bộ nhớ này có thể được truy cập thông qua tên biến A và B. In the next step, the variables are added, and the result is stored in a third variable called C. Trong bước tiếp theo, các biến được cộng lại và kết quả được lưu trong một biến thứ ba có tên là C. Finally, the value in this variable is displayed. Cuối cùng, giá trị trong biến này được hiển thị.

While some languages allow the operating system to delete the contents of memory locations and allocate this memory for reuse, Trong khi một số ngôn ngữ cho phép hệ điều hành xóa nội dung của các vị trí bộ nhớ và phân bổ lại bộ nhớ này để sử dụng lại, other languages such as C require the programmer to clear unused memory locations through code. các ngôn ngữ khác như C yêu cầu lập trình viên phải xóa các vị trí bộ nhớ không sử dụng thông qua mã. In both these cases, it is the operating system that takes care of allocating and deallocating memory. Trong cả hai trường hợp này, chính hệ điều hành sẽ lo liệu việc phân bổ và giải phóng bộ nhớ.

2.2 Constants

In case of variables, the values stored in them need not be the same throughout its lifetime. Trong trường hợp các biến, các giá trị được lưu trữ trong chúng không cần phải giống nhau trong suốt vòng đời của chúng. The statements within this scope block can access the value of this variable. Các câu lệnh trong khối phạm vi này có thể truy cập giá trị của biến này. They even can change the value of the variables. Chúng thậm chí có thể thay đổi giá trị của các biến. There is need for certain items whose value can never be changed. Có những mục cần thiết mà giá trị của chúng không bao giờ có thể thay đổi.

A constant is a value whose worth never changes. Một hằng số là một giá trị mà giá trị của nó không bao giờ thay đổi. For example, 5 is a constant, whose mathematical value is always 5 and can never be changed by anybody. Ví dụ, 5 là một hằng số, có giá trị toán học luôn là 5 và không thể bị thay đổi bởi bất kỳ ai. Similarly, ‘Black’ is a constant that represents black color alone. Tương tự, ‘Black’ là một hằng số chỉ đại diện cho màu đen. While 5 is termed as numeric constant, ‘Black’ is termed as string constant. Trong khi 5 được gọi là hằng số số, ‘Black’ được gọi là hằng số chuỗi.

2.3 Identifier

The names of variables, functions, labels, and various other user-defined objects are called identifiers. Tên của các biến, hàm, nhãn và nhiều đối tượng do người dùng định nghĩa khác được gọi là các định danh. These identifiers can contain one or more characters. Các định danh này có thể chứa một hoặc nhiều ký tự. It is compulsory that the first character of the identifier is a letter or an underscore (). **Điều bắt buộc là ký tự đầu tiên của định danh phải là một chữ cái hoặc một dấu gạch dưới ().** The subsequent characters can be alphabets, numbers, or underscores. Các ký tự tiếp theo có thể là chữ cái, số hoặc dấu gạch dưới.

Some correct identifier names are `arena`, `s_count`, `marks40`, and `class_one`. Examples of wrong identifier names are `1sttest`, `oh!god`, and `start... end`.

Identifiers can be of any convenient length, but the number of characters in a variable that are recognized by a compiler varies from compiler to compiler. Các định danh có thể có độ dài thuận tiện, nhưng số ký tự trong một biến mà một trình biên dịch nhận diện sẽ khác nhau tùy vào từng trình biên dịch. For example, if a given compiler recognizes the first 31 significant digits of an identifier name, then the following will appear the same to it. Ví dụ, nếu một trình biên dịch nhất định nhận diện 31 chữ số quan trọng đầu tiên của một tên định danh, thì các tên sau đây sẽ được xem là giống nhau.

Concepts

```
This is a testing variable .... testing
This is a testing variable .... testing ... testing
```

Identifiers in C are case sensitive, that is, **arena** is different from **ARENA**.

2.3.1 Guidelines for Specifying Identifier Names

The rules for naming variables are different for each programming language. Các quy tắc đặt tên biến thì khác nhau cho mỗi ngôn ngữ lập trình. However, some conventions that are typically followed are: Tuy nhiên, một số quy ước thường được tuân theo là:

- Variable names must begin with an alphabet. Tên biến phải bắt đầu bằng một chữ cái.
- The first character may be followed by a sequence of letters or digits and can also include a special character like Ký tự đầu tiên có thể được theo sau bởi một chuỗi các chữ cái hoặc số và cũng có thể bao gồm một ký tự đặc biệt như.
- Avoid using the letter O in places where it can be confused with the number 0, Tránh sử dụng chữ O ở những chỗ có thể bị nhầm với số 0, and similarly, the lowercase letter L can be mistaken for the number 1. và tương tự, chữ cái thường l có thể bị nhầm với số 1
- Proper names should be avoided while naming variables. Nên tránh sử dụng tên riêng khi đặt tên cho các biến.
- Typically, uppercase and lowercase letters are treated as different, Thông thường, chữ hoa và chữ thường được coi là khác nhau, i.e., variables ADD, add, and Add are all different. tức là, các biến ADD, add và Add đều là khác nhau.
- As case-sensitivity considerations vary with programming languages, Vì các quy tắc phân biệt chữ hoa chữ thường khác nhau giữa các ngôn ngữ lập trình, it is advisable to maintain a standard way of naming variables. nên tốt nhất là duy trì một cách đặt tên biến theo tiêu chuẩn
- A variable name should be meaningful and descriptive; it should describe the kind of data it holds. Tên biến nên có ý nghĩa và mô tả; nó nên mô tả loại dữ liệu mà nó chứa. For example, if the sum of two numbers is to be found, Ví dụ, nếu tổng của hai số cần được tìm, the variable storing the result may be called sum. biến lưu trữ kết quả có thể được gọi là sum. Naming it s or ab12 is not a good idea. Việc đặt tên là s hoặc ab12 không phải là một ý hay.

2.3.2 Keywords

All languages reserve certain words for their internal use. Tất cả các ngôn ngữ đều dành riêng một số từ cho mục đích nội bộ của chúng. These words hold a special meaning within the context of the particular language, and are referred to as 'keywords'. Những từ này có một ý nghĩa đặc biệt trong ngữ cảnh của ngôn ngữ đó và được gọi là 'từ khóa'. While naming variables, we need to ensure that we do not use one of these keywords as a variable name. Khi đặt tên cho các biến, chúng ta cần đảm bảo rằng không sử dụng một trong những từ khóa này làm tên biến.

For example, all the data types are reserved as keywords. Ví dụ, tất cả các kiểu dữ liệu đều được dành riêng làm từ khóa.

Hence, naming a variable int will generate an error, Vì vậy, việc đặt tên biến là int sẽ tạo ra lỗi, but naming a variable integer will not. nhưng việc đặt tên biến là integer sẽ không gây ra lỗi.

Some programming languages require the programmer to specify the name of the variable as well as the type of data that is to be stored in it, before actually using a variable. This step is referred to as

'variable declaration'.

2.4 Data types

Different types of data that can be stored in variables are :

- **Numbers**
  - Whole numbers.

    Example, 10 or 178993455
  - Real numbers.

    Example, 15.22 or 15463452.25
- **Names**

  Example, John
- **Logical values**

  Example, Y or N

As the data that stored in variables is of different types, it requires different amounts of memory.

The amount of memory to be assigned to a variable depends on the data type stored in it.

To assign memory for a particular piece of data, we have to declare a variable of a particular data type.

The term, declaring a variable, means that some memory is assigned to it. Thuật ngữ khai báo một biến có nghĩa là một số bộ nhớ được gán cho nó. This portion of memory is then referred to by the variable name. Phần bộ nhớ này sau đó được tham chiếu bởi tên biến. The amount of memory assigned by the operating system depends on the type of the data that is going to be stored in the variable. Lượng bộ nhớ được phân bổ bởi hệ điều hành phụ thuộc vào loại dữ liệu sẽ được lưu trữ trong biến. A data type, therefore, describes the kind of data that will fit into a var. Vì vậy, kiểu dữ liệu mô tả loại dữ liệu nào sẽ phù hợp với một biến.

The general form of declaring a variable is:

Concepts

```
Data type (variable name)
```

Data types, which are commonly used in programming tools, can be classified as:

1. Numeric data type - stores numeric value
2. Alphanumeric data type - stores descriptive information

These data types can have different names in different programming languages. Các kiểu dữ liệu này có thể có tên khác nhau trong các ngôn ngữ lập trình khác nhau. For example, a numeric data type is called int in C language while Visual Basic refers to it as integer. Ví dụ, một kiểu dữ liệu số được gọi là int trong ngôn ngữ C trong khi Visual Basic gọi nó là integer. Similarly, an alphanumeric data type is named char in C while in Visual Basic it is named string. Tương tự, một kiểu dữ liệu alphanumeric được đặt tên là char trong C trong khi trong Visual Basic nó được gọi là string. In any case, the data stored is the same. Trong bất kỳ trường hợp nào, dữ liệu được lưu trữ là giống nhau. The only difference is that the variables used in a tool have to be declared with the name of the data type supported by that tool. Sự khác biệt duy nhất là các biến được sử dụng trong một công cụ phải được khai báo với tên của kiểu dữ liệu được công cụ đó hỗ trợ.

C has five basic data types. All other data types are based upon one of these types. The five basic types are:

- `int` is an integer, typically representing the natural size of **integers**.
- `float` and `double` are used for floating point numbers. `float` occupies 4 bytes and can hold numbers with six digits of precision, whereas `double` occupy eight bytes and can hold numbers with ten digits of precision.
- `char` type occupies one byte long and is capable of holding one **character**.
- `void` is typically used to declare a function as returning **no value**. This will be clearer after the session on functions.

The size and range of these types vary with each processor type and with every implementation of the C compiler.

Note: Floating point numbers are used to represent values, which require a decimal point precision.

- **Type `int`**

  The data type, which stores numeric data, is one of the basic data types in any programming language. It consists of a sequence of one or more digits.

  For example in C, to store an integer value in a variable called ‘num’, the declaration would be as follows:

  ```
  int num;
  ```

  The variable ‘num’ cannot store any other type of data like “Alan” or “abc”. This numeric data type

allows integers in the range **-32768** to **32767** to be stored. The OS allocates 16 bits (2 bytes) to a variable that is declared to be of type `int`. Examples: **12322**, **0**, **-232**.

If we assign the value **12322** to **num**, then **num** is a float variable and **12322** is a float constant.

- **Type `float`**

A variable of type `float` is used for storing values containing decimal places. The compiler differentiates between the data types, `float` and `int`.

The main difference between the two is that `int` data type includes only whole numbers, whereas `float` data type can hold either whole or fractional numbers.

For example, in C, to store a `float` value in a variable called 'num', the declaration would be as follows:

```
float num;
```

Variable declared to be of type `float` can store decimal values with a precision of upto 6 digits. The variable is allocated 32 bits (4 bytes) of memory. Examples: **23.05**, **56.5**, **32**.

If we assign the value **23.5** to **num**, then **num** is a float variable and **23.5** is a float constant.

- **Type `double`**

The type `double` is used when the value to be stored exceeds the size limit of type `float`. Variables of the type `double` can roughly store twice the number of digits as `float` type.

The precise number of digits the `float` or `double` data type can store depends upon the particular computer system.

Numbers stored in the data type `float` or `double` are treated identically by the system in terms of calculation. However, using the type `float` saves memory as it takes only half the space as a `double` would.

`double` data type, allows a greater degree of precision (upto 10 digits). A variable declared of type `double` is allocated 64 bits (8 bytes).

For example in C, to store a `double` value in a variable called 'num', the declaration would be as follows:

```
double num;
```

Concepts

If we assign the value **23.34232324** to **num**, then **num** is a double variable and **23.34232324** is a double constant.

- **Type char**

The data type char is used to store a single character of information.

A char type can store a single character enclosed within two single quotation marks. Some valid examples of char types are '**a**', '**m**', '**$**' '**%**'.

It is also possible to store digits as characters by enclosing them within single quotes. Cũng có thể lưu trữ các chữ số dưới dạng ký tự bằng cách đặt chúng trong dấu nháy đơn. These should not be confused with the numeric values. Những ký tự này không nên bị nhầm lẫn với các giá trị số. For example, '1', '5' and '9' are not to be confused with the numbers 1, 5 and 9. Ví dụ, '1', '5' và '9' không nên bị nhầm lẫn với các số 1, 5 và 9.

Consider the statements of C code provided below.

```
char gender;
gender='M';
```

The first line declares the variable 'gender' of data type char. Dòng đầu tiên khai báo biến 'gender' có kiểu dữ liệu char. The second line stores an initial value of 'M' in it. Dòng thứ hai lưu trữ một giá trị khởi tạo là 'M' trong đó. The variable gender is a character variable and 'M' is a character constant. Biến gender là một biến ký tự và 'M' là một hằng số ký tự. The variable is allocated 8 bits (one byte). Biến này được phân bổ 8 bit (một byte)

- **Type void**

C has a special data type called void. C có một kiểu dữ liệu đặc biệt gọi là void. This data type indicates to the C compiler that there is no data of any type. Kiểu dữ liệu này cho biết với trình biên dịch C rằng không có dữ liệu của bất kỳ loại nào. In C, functions return data of some type. Trong C, các hàm trả về dữ liệu của một kiểu nào đó. However, when a function has nothing to return, the void data type can be used to indicate this. Tuy nhiên, khi một hàm không có gì để trả về, kiểu dữ liệu void có thể được sử dụng để chỉ định điều này.

2.4.1 Basic and Derived Data types

The four (char, int, float and double) data types that we have discussed above are used for actual data representation in the computer's memory. These data types can be modified to fit various situations more precisely. The resulting data types are said to have been derived from these basic types.

A modifier is used to alter the base type to fit various situations. Except for void, all other data types can have modifiers preceding them. Modifiers used with C are signed, unsigned, long and short. All of these can be applied to character and integer data type. The long modifier can also be applied to double.

Some of the modifiers are:

1. `unsigned`
2. `long`
3. `short`

To declare a variable of a derived type, we need to precede the usual variable declaration with one of the modifiers. Để khai báo một biến có kiểu dữ liệu kế thừa, chúng ta cần đặt trước khai báo biến thông thường với một trong các từ sửa đổi. A detailed explanation of these modifiers and how they can be used is provided below. Một giải thích chi tiết về các từ sửa đổi này và cách chúng có thể được sử dụng sẽ được cung cấp bên dưới.

The `signed` and `unsigned` Types

- Default integer declaration assumes a signed number. Khai báo số nguyên mặc định giả định một số có dấu. The most important use of signed is to modify the char data type, Sự sử dụng quan trọng nhất của signed là để sửa đổi kiểu dữ liệu char, where char is unsigned by default. trong đó char mặc định là không có dấu.

The unsigned type specifies that a variable can take only positive values. Kiểu không có dấu xác định rằng một biến chỉ có thể nhận các giá trị dương. This modifier may be used with the int and float data types. Bộ sửa đổi này có thể được sử dụng với các kiểu dữ liệu int và float. Unsigned can be applied to floating data types in some cases, but this reduces Kiểu không có dấu có thể được áp dụng cho các kiểu dữ liệu số thực trong một số trường hợp, nhưng điều này làm giảm the portability of the code. tính di động của mã.

By prefixing the `int` type with the word `unsigned`, the range of positive numbers can be doubled.

Consider the statements of C code provided below, which declares a variable of type `unsigned int`, and initializes this variable to **`23123`**.

```
unsigned int varNum;
varNum = 23123;
```

Note that the space allocated to this type of variable remains the same. Lưu ý rằng không gian được phân bổ cho kiểu biến này vẫn giữ nguyên. That is, the variable varNum is allocated 2 bytes since it is an int. Có nghĩa là, biến varNum được phân bổ 2 byte vì nó là một int. However, the values that an unsigned int supports range from 0 to 65535, Tuy nhiên, các giá trị mà một unsigned int hỗ trợ nằm trong khoảng từ 0 đến 65535, instead of –32768 to 32767 that an int supports. thay vì từ –32768 đến 32767 mà một int hỗ trợ. By default, an int is a signed int. Theo mặc định, một int là một int có dấu.

The `long` and `short` Types

- They are used when an integer of length shorter or longer than its normal length is required. Chúng được sử dụng khi cần một số nguyên có độ dài ngắn hơn hoặc dài hơn độ dài bình thường của nó. A short modifier is applied to the data type when a length shorter than the normal integer length is sufficient, Một bộ sửa đổi ngắn được áp dụng cho kiểu dữ liệu khi một độ dài ngắn hơn độ dài số nguyên bình thường là đủ, and a long modifier is used when a length of more than the normal integer length is required. và một bộ sửa đổi dài được sử dụng khi một độ dài lớn hơn độ dài số nguyên bình thường là cần thiết.

The `short` modifier is used with the `int` data type. It modifies the `int` data type so that it occupies

less memory place. Therefore, while a variable of type `int` occupies 16 bits (2 bytes), a variable of type `short int` (or simply `short`), occupies 8 bits (1 byte), and allows numbers in the range **`-128`** to **`127`**.

The `long` modifier is used to accommodate a wider range of numbers. It can be used with the `int` as well as the `double` data types. When used with the `int` data type, the variable supports numbers between **`-2,147,483,647`** and **`2,147,483,647`**, and occupies 32 bits (4 bytes). Similarly, a 'long double' type of variable occupies 128 bits (16 bytes).

A `long int` variable is declared as follows:

```
long int varNum;
```

It can also be declared simply as `long varNum` as well. A long integer can be declared as `long int` or just `long`. Similarly, a short integer can be declared as `short int` or `short`.

Given below is a table, which shows the range of values for the different data types and the number of bits taken. This is based on the ANSI standard.

Type	Approximate Size in Bits	Minimal Range
char	8	-128 to 127
unsigned	8	0 to 255
signed char	8	-128 to 127
int	16	-32,768 to 32,767
unsigned int	16	0 to 65,535
signed int	16	Same as int
short int	16	Same as int
unsigned short int	8	0 to 65, 535
signed short int	8	Same as short int
signed short int	8	Same as short int
long int	32	-2,147,483,647 to 2,147,483,647
signed long int	32	0 to 4,294,967,295
unsigned long int	32	0 to 4,294,967,295
float	32	Six digits of precision
double	64	Ten digits of precision
long double	128	Ten digits of precision

Table 2.1: Data Types and their range

The following example shows declarations of the above-discussed types.

Example 2:

```
main ()
{
    char abc; /*abc of type character */
    int xyz; /*xyz of type integer */
    float length; /*length of type float */
    double area; /* area of type double */
    long liteyrs; /*liteyrs of type long int */
    short arm; /*arm of type short integer*/
}
```

Let us take the same example which we dealt with in the last session to add two numbers and display their sum. Hãy lấy ví dụ giống như chúng ta đã thảo luận trong phiên trước để cộng hai số và hiển thị tổng của chúng. The pseudo code for the same is given in example 3. Mã giả cho điều này được đưa ra trong ví dụ 3.

Example 3:

```
BEGIN
    INPUT A, B
    C = A + B
    DISPLAY C
END
```

In this example, the values for two variables, A and B, are accepted. Trong ví dụ này, các giá trị cho hai biến A và B được chấp nhận. The values are added, and the sum is stored in C using the statement C = A + B. Các giá trị được cộng lại, và tổng được lưu trữ trong C bằng cách sử dụng câu lệnh C = A + B. In this statement, A and B are variables, and the symbol + is called an operator. Trong câu lệnh này, A và B là các biến, và ký hiệu + được gọi là toán tử. Let us discuss the various arithmetic operators available in C in the following section. Hãy cùng thảo luận về các toán tử số học khác nhau có sẵn trong C ở phần tiếp theo. However, there are other types of operators available in C which will be covered in the next session. Tuy nhiên, còn có các loại toán tử khác có sẵn trong C mà sẽ được đề cập trong phiên tiếp theo.

2.5 Arithmetic Operators

Arithmetic operators are used to perform numerical operations. They are divided into two classes: **unary** and **binary** arithmetic operators.

Table 2.2 lists the arithmetic operators and their action.

Unary Operators	Action	Binary Operators	Action
-	Unary minus	+	Addition
+ +	Increment	-	Subtraction
--	Decrement	*	Multiplication
		%	Modulus

Concepts

Unary Operators	Action	Binary Operators	Action
		/	Division
		^	Exponentiation

Table 2.2: Arithmetic operators and their action

➢ **Binary Operators**

In C, the binary operators work as they do in other languages. Trong C, các toán tử nhị phân hoạt động giống như trong các ngôn ngữ khác. Operators like +, -, * and / can be applied to almost any built-in data type allowed by C. Các toán tử như +, -, * và / có thể được áp dụng cho hầu hết các kiểu dữ liệu tích hợp mà C cho phép. When / is applied to an integer or character, any remainder will be truncated. Khi / được áp dụng cho một số nguyên hoặc ký tự, bất kỳ phần dư nào sẽ bị cắt ngắn. For example, 5/2 will equal to 2 in integer division. Ví dụ, 5/2 sẽ bằng 2 trong phép chia số nguyên. The modulus operator (%) gives the remainder of an integer division. Toán tử modulus (%) cho phần dư của phép chia số nguyên. For example, 5%2 will give 1. Ví dụ, 5%2 sẽ cho 1. However, % cannot be used with floating point types. Tuy nhiên, % không thể được sử dụng với các kiểu số thực.

Let us consider an example for the exponentiation operator.

```
9 ^ 2
```

Here, 9 is the base and 2 is the exponent.

The number to the left of '^' is known as the base and the number to the right of '^' is known as the exponent.

The result of `9^2` evaluates to is `9 * 9 = 81`.

Another example is:

```
5 ^ 3
```

This effectively means:

```
5 * 5 * 5
```

Thus, `5 ^ 3 = 5 * 5 * 5 = 125`.

Note: The programming languages like Basic, supports the ^ operator to carry out exponentiation. However, ANSI C does not support the symbol, ^, for exponentiation. The alternate way to calculate exponentiation in C is to use `pow()` function defined in `math.h`. The syntax to carry out the same is as follows:

```
#include<math.h>
void main(void)
{
    ...
    /* The following function will calculate x to the power y */
    z = pow(x,y);
    ...
}
```

The following example shows all the binary operators used in C. Note that the functions `printf()` and `getchar()` are not yet covered. We will be discussing the same in the forthcoming sessions.

Example 4:

```
#include<stdio.h>
main()
{
    int x,y;
    x = 5;
    y = 2;
    printf("The integers are : %d & %d\n", x, y);
    printf("The addition gives : %d\n", x+y);
    printf("The subtraction gives : %d\n", x-y);
    printf("The multiplication gives : %d\n", x*y);
    printf("The division gives : %d\n", x/y);
    printf("The modulus gives : %d\n", x%y);
    getchar();
}
```

The above code will produce the following output:

```
The integers are : 5 & 2
The addition gives : 7
The subtraction gives : 3
The multiplication gives : 10
The division gives : 2
The modulus gives : 1
```

➢ Unary Arithmetic Operators

The unary arithmetic operators are unary minus (–), increment operator (++) and decrement operator (––).

• Unary Minus (–)

The symbol is the same as used for binary subtraction. Unary minus is used to indicate or change the algebraic sign of a value. For Example,

```
a = -75;

b = -a;
```

The above assignments result in `a` being assigned the value **`-75`** and **`b`** being assigned the value **`75`** (**`-(-75)`**). The minus sign used in this way is called the unary operator because it takes just one operand.

Strictly speaking, there is no unary + in C. Hence, an assignment statement like,

```
invld_pls = +50;
```

where, **`invld_pls`** is an integer variable, is not valid in standard C. However, many compilers do not object to such usage.

• Increment and Decrement Operators

C includes two useful operators that are generally not found in other computer languages. These are the increment operator (++) and the decrement operator (––). The operator ++ adds 1 to its operand, whereas the operator –– subtracts 1 from its operand.

To be precise,

```
x = x + 1;
```

can be written as

```
x++;
```

and

```
x = x - 1;
```

can be written as

```
x--;
```

Both these operators may either precede or follow the operand, i.e.

```
x = x + 1;
```

can be represented as

```
x++ or ++x;
```

Similar operations hold true for -- operator.

The difference between pre-fixing and post-fixing the operator is useful when it is used in an expression. Sự khác biệt giữa việc tiền tố và hậu tố toán tử là hữu ích khi nó được sử dụng trong một biểu thức. When the operator precedes the operand, C performs the increment or decrement operation before using the value of the operand. Khi toán tử đứng trước toán hạng, C thực hiện phép tăng hoặc giảm trước khi sử dụng giá trị của toán hạng. This is known as pre-fixing. Điều này được gọi là tiền tố. If the operator follows the operand, the value of the operand is used before incrementing or decrementing. Nếu toán tử đứng sau toán hạng, giá trị của toán hạng sẽ được sử dụng trước khi tăng hoặc giảm nó.

Consider the following:

```
a = 10;
b = 5;
c = a * b++;
```

In the above expression, the current value of **b** is used for the product and then the value of **b** is incremented. That is, **c** is assigned **50(a*b)** and then the value of **b** is incremented to **6**.

If, however, the above expression was

```
c = a * ++b;
```

the value stored in **c** would be **60**, because **b** would first be incremented by **1** and then the value stored in **c(10*6)**.

In a context where the incrementing or the decrementing effect alone is required, the operators can be used either as postfix or prefix.

Most C compilers produce very fast, efficient object code for increment and decrement operations. Hầu hết các trình biên dịch C tạo ra mã đối tượng rất nhanh và hiệu quả cho các phép tăng và giảm. This code is better than that generated by using the equivalent assignment statement. Mã này tốt hơn mã được tạo ra bằng cách sử dụng câu lệnh gán tương đương. So, increment and decrement operators should be used whenever Vì vậy, các toán tử tăng và giảm nên được sử dụng bất cứ khi nào co the

Concepts

Summary

- Most often, an application needs to handle data; it needs some place where this data can be temporarily stored. This 'place' where this data is stored is called the memory.
- Modern day languages enable us to use symbolic names known as variables, to refer to the memory location where a particular value is to be stored.
- There is no limit to the number of memory locations that a program can use.
- A constant is a value whose worth never changes.
- The names of variables, functions, labels, and various other user-defined objects are called identifiers.
- All languages reserve certain words for their internal use. They are called keywords.
- The main data types of C are character, integer, float, double float and void.
- Modifiers are used to alter the basic data types so as to fit into various situations. Unsigned, short and long are the three modifiers available in C.
- C supports two types of Arithmetic operators: Unary and Binary.
- Increment(++) and decrement(– –) are unary operators acting only on numeric variables.
- Arithmetic binary operations are + - * / % which act only on numeric constants, variables or expressions.
- The modulus operator % acts only on integers and results in remainder after integer division.

Check Your Progress

1. C is case sensitive. **(True / False)**

2. The number 10 is a _____.

3. The first character of the identifier can be a number. **(True / False)**

4. Using the type _____ saves memory as it takes only half the space as a _____ would.

5. The _____ data type is used to indicate the C compiler that no value is being returned.

6. _____ and _____ are the two classes of arithmetic operators.

   A. Bitwise & and | B. Unary and Binary

   C. Logical AND D. None of the above

7. The unary arithmetic operators are ___and ___ .

   A. ++ and – – B. % and ^

   C. ^ and $ D. None of the above

Try It Yourself

1. Match the following:

Column A	Column B
8	
10.34	Invalid Identifier Names
A B C	Integer Constants
abc	Character Constants
23	Double
12112134.86868686886	Floating Point Numbers
_A1	Valid Identifier Names
$abc	
'A'	

Hint: Multiple items in Column A can map to a single element in Column B.

2. What will be the value of the variables at the end in each of the following code statements:

a.
```
int a=4^4
```

b.
```
int a=23.34
```

c.
```
a = 10

b = a + a++
```

d.
```
a=-5

b=-a
```

Session
3 Variables and Data Types (Lab)

Lab Guide

Objectives

At the end of this session, you will be able to:

- *Use variable, data type and arithmetic expression*

Part I – For the first 1 Hour and 30 Minutes :

3.1 Variables

As we already know, variables are names given to locations in the memory of a computer which may store different values at different instances of time. Như chúng ta đã biết, biến là tên được gán cho các vị trí trong bộ nhớ của máy tính có thể lưu trữ các giá trị khác nhau vào những thời điểm khác nhau. In this session, let us focus on how to create and use variables. Trong phiên này, hãy cùng chúng ta tập trung vào cách tạo và sử dụng biến.

3.1.1 Creating a variable

Variable creation involves the data type and the logical name for that variable. For example,

```
int currentVal;
```

In the example above, the name of variable is **`currentVal`** which is of integer data-type.

3.2 Data Type

A data type defines what kind of value will be stored by a particular variable. For example,

```
int currentVal;
```

In the example above, **`int`** specifies that **`currentVal`** variable will store integer value.

3.3 Arithmetic Expression

A C arithmetic expression consists of a variable name on the left-hand side of "=" and variable names and constants on the right-hand side of "=". Một biểu thức số học trong C bao gồm một tên biến ở phía bên trái của "=" và các tên biến cùng với các hằng số ở phía bên phải của "=". The variables and constants appearing on the right-hand side of "=" are connected by arithmetic operators like +, -, *, and /. *Các biến và hằng số xuất hiện ở phía bên phải của "=" được liên kết bởi các toán tử số học như +, -, , và /. For example, Ví dụ,

```
delta = alpha * beta / gamma + 3.2 * 2 / 5;
```

Now, Let us look at a complete program to calculate the simple interest.

1. **Invoke the editor in which you can type the C program.**
2. **Create a new file.**
3. **Type the following code:**

```
#include<stdio.h>
void main()
{
    int principal, period;
    float rate, si;
    principal = 1000;
    period = 3;
    rate = 8.5;
    si = principal * period * rate / 100;
    printf("%f", si);
}
```

To see the output, follow these steps:

4. **Save the file with the name myprogram1.C**
5. **Compile the file, myprogram1.C**
6. **Execute the program, myprogram1.C**
7. **Return to the editor.**

The sample output of the above program is shown in Figure 3.1.

Figure 3.1: Output of myprogram1.C

1. **Create a new file.**

2. **Type the following code :**

```
#include<stdio.h>
void main()
{
    int a, b, c, sum;
    printf("\n Enter any three numbers: ");
    scanf("%d %d %d", &a, &b, &c);
    sum = a + b + c;
    printf("\n Sum = %d", sum);
}
```

3. **Save the file with the name myprogramII.C**

4. **Compile the file, myprogramII.C**

5. **Execute the program, myprogramII.C**

6. **Return to the editor.**

The sample output of the above program will be as shown in Figure 3.2.

Figure 3.2: Output I of myprogramII.C

Part II – For the next 30 Minutes:

1. Write a C program that accepts a number and square the number.

   To do this,

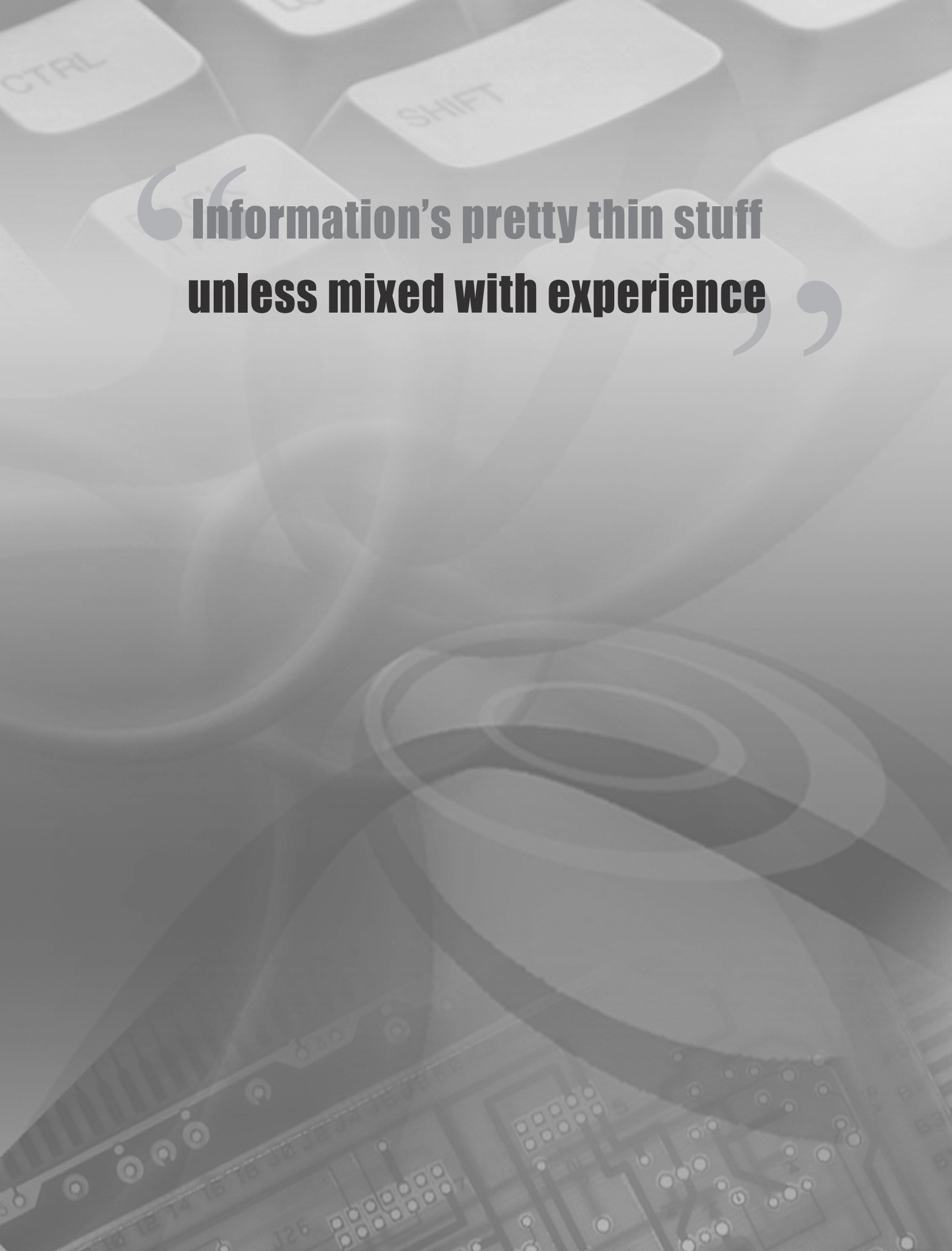
   a. Accept the number.

   b. Multiply the number with itself and display the square.

Try It Yourself

1. Write a C program to find the area and perimeter of a circle.

2. Write a C program that accepts the salary and age from the user and displays the same on the screen as output.



“Information’s pretty thin stuff
unless mixed with experience”

Session

4 Operators and Expressions

Concepts

Objectives

At the end of this session, you will be able to:

- *Explain Assignment Operators*
- *Understand Arithmetic Expressions*
- *Explain Relational and Logical Operators*
- *Understand Bitwise Logical Operators and Expressions*
- *Explain Casts*
- *Understand the Precedence of Operators*

Introduction

C has a very rich set of operators. C có một bộ toán tử rất phong phú. Operators are the tools that manipulate data. Các toán tử là công cụ để thao tác dữ liệu. An operator is a symbol, which represents some particular operation to be performed on the data. Một toán tử là một ký hiệu, đại diện cho một phép toán cụ thể nào đó được thực hiện trên dữ liệu. C defines four classes of operators: arithmetic, relational, logical, and bitwise. C định nghĩa bốn lớp toán tử: toán học, so sánh, logic và theo bit. In addition, C has some special operators for special tasks. Ngoài ra, C còn có một số toán tử đặc biệt cho các tác vụ đặc biệt.

Operators operate on constants or variables, which are called operands. Các toán tử hoạt động trên các hằng số hoặc biến, được gọi là toán hạng. Variables have already been discussed in the previous session. Các biến đã được thảo luận trong phiên trước. Constants are fixed values that the program cannot alter. Các hằng số là các giá trị cố định mà chương trình không thể thay đổi. C constants can be of any of the basic data types. Các hằng số trong C có thể thuộc bất kỳ kiểu dữ liệu cơ bản nào. Operators can be classified as either unary, binary, or ternary operators. Các toán tử có thể được phân loại thành toán tử đơn, toán tử nhị phân hoặc toán tử tam. Unary operators act on only one data element, Toán tử đơn hoạt động trên chỉ một phần tử dữ liệu, binary operators act on two data elements, toán tử nhị phân hoạt động trên hai phần tử dữ liệu, and ternary operators operate on three data elements. và toán tử tam hoạt động trên ba phần tử dữ liệu.

```
c = a + b;
```

Here **a**, **b**, **c** are the operands and '=' and '+' are the operators.

4.1 Expressions

An expression can be any combination of operators and operands. Một biểu thức có thể là bất kỳ sự kết hợp nào của các toán tử và toán hạng. Operators perform operations like addition, subtraction, comparison, etc. Các toán tử thực hiện các phép toán như cộng, trừ, so sánh, v.v. Operands are the variables or values on which the operations are performed. Toán hạng là các biến hoặc giá trị mà trên đó các phép toán được thực hiện. For example, in a + b, a and b are operands and + is the operator. Ví dụ, trong a + b, a và b là các toán hạng và + là toán tử. The whole thing together is an expression. Toàn bộ kết hợp này được gọi là một biểu thức.

Session 4

Concepts

During the execution of the program, the actual value of the variable (if any) is used along with the constants that are present in the expression. Trong quá trình thực thi chương trình, giá trị thực tế của biến (nếu có) được sử dụng cùng với các hằng số có trong biểu thức. Evaluation takes place with the help of operators. Việc đánh giá diễn ra với sự trợ giúp của các toán tử. Thus, every C expression has a value. Vì vậy, mỗi biểu thức trong C đều có một giá trị.

The following are examples of expressions:

```
2
x
3 + 7
2 × y + 5
2 + 6 × (4 - 2)
z + 3 × (8 - z)
```

Roland weighs 70 kilograms, and Mark weighs **k** kilograms. Write an expression for their combined (or total) weight. The combined weight in kilograms of these two people is the sum of their weights, which is **70 + k.**

Evaluate the expression `4 × z + 12` when `z = 15`.

We replace each occurrence of **z** with the number 15, and simplify using the usual rules: parentheses (or brackets) first, then exponents, multiplication and division, then addition and subtraction.

```
4 × z + 12 becomes
4 × 15 + 12 = (multiplication comes before addition)
60 + 12 =
72
```

The Assignment Operator

Before looking into other operators, it is essential to discuss the assignment operator (=). Trước khi xem xét các toán tử khác, điều cần thiết là thảo luận về toán tử gán (=). It is the most common operator in any language and well known to everyone. Đây là toán tử phổ biến nhất trong bất kỳ ngôn ngữ nào và được mọi người biết đến. In C, the assignment operator can be used with any valid C expression. Trong C, toán tử gán có thể được sử dụng với bất kỳ biểu thức C hợp lệ nào. The general form of the assignment operator is: Công thức tổng quát của toán tử gán là:

```
variable_name = expression;
```

Multiple Assignment

Many variables can be assigned the same value in a single statement. This is done by using the multiple assignment syntax. For example:

```
a = b = c =10;
```

The above line of code assigns the value 10 to **a, b**, and **c**. However this cannot be done at the time of declaration of the variables. For example,

```
int a = int b = int c = 0;
```

This above statement gives an error because the syntax for multiple assignments is wrong here.

Arithmetic Expressions

The operations are carried out in a specific (or a particular) order, to arrive at the final value. This order is known as order of precedence (discussed later).

Mathematical expressions can be expressed in C using the arithmetic operators with numeric and character operands. Such expressions are known as **Arithmetic Expressions**. Examples of arithmetic expressions are:

```
a * (b+c/d)/22;
++i % 7;
5 + (c = 3+8);
```

As seen above, operands can be constants, variables, or a combination of the two. Như đã thấy ở trên, toán hạng có thể là các hằng số, biến hoặc sự kết hợp của cả hai. Also, an expression can be a combination of several smaller sub-expressions. Ngoài ra, một biểu thức có thể là sự kết hợp của nhiều biểu thức con nhỏ hơn. For instance, in the first expression, c/d is a sub-expression, and in the third expression c = 3 + 8 is also a sub-expression. Chẳng hạn, trong biểu thức đầu tiên, c/d là một biểu thức con, và trong biểu thức thứ ba, c = 3 + 8 cũng là một biểu thức con.

4.2 Relational Operators and Expressions

Relational operators are used to test the relationship between two variables, or between a variable and a constant. For example, the test for the larger of the two numbers, **a** and **b**, is made by means of the greater than (>) sign between the two operands **a** and **b (a > b)**.

In C, **true** is any value other than zero, and false is zero. Expressions that use relational operators, return 0 for false and 1 for true. For example, consider the following expression:

```
a == 14;
```

This expression checks for equality, in the sense that it checks whether the value contained in the variable **a** is equal to 14 or not. The value of this expression is said to be `0` (`false`) if **a** has a value that is not equal to 14 and `1` (`true`) if it is 14.

The relational operators and the operation they perform are listed in Table 4.1 below.

Operator	Relational Operators Action
>	Greater than
> =	Greater than or equal
<	Less than
< =	Less than or equal
= =	Equal
!=	Not equal

Table 4.1: Relational operators and their action

4.3 Logical Operators and expressions

Logical operators are symbols that are used to combine or negate expressions containing relational operators.

Expressions that use logical operators return `0` for `false` and `1` for `true`.

The logical operators and the operation they perform are listed in Table 4.2 below.

Operator	Logical Operators Action
&&	A N D
||	O R
!	NOT

Table 4.2: Logical operators and their action

As for the explanation of the logical operators, consider the following. Any operator that uses two characters as a symbol should have no space between the two characters, that is == is not correct if it is written as = =.

Suppose a program has to perform certain steps if the conditions **`a < 10`** and **`b == 7`** are satisfied. This condition is coded using the relational operator in conjunction with the logical operator `AND`. This `AND` operator is represented by `&&`, and the condition will be written as:

```
(a < 10) && (b == 7);
```

Similarly the `OR` is operator used to check whether one of the conditions is true. It is represented by two consecutive pipe signs (`||`). If the above example is considered in such a way that the steps in the program have to be performed if either of the statements are true then the condition will be coded as:

```
(a < 10) || (b == 7);
```

The third logical operator NOT is represented by a single exclamation mark (!). This operator reverses the true value of the expression. For example, if a program has to do some steps if a variable **s** is less than 10, the condition can be written as

```
(s < 10);
```

as well as

```
(! (s >= 10)) /* s is not equal to or more than 10 */
```

Both relational and logical operators are lower in precedence than the arithmetic operators. For example, **5 > 4 + 3** is evaluated as if it is written as **5 > (4 + 3)**, that is, **4+3** will be evaluated first and then the relational operation will be carried out. The result for this will be false, that is 0 will be returned.

The following statement

```
printf("%d", 5> 4 + 3);
```

will give

```
0
```

as 5 is less than 7 (4 + 3).

4.4 Bitwise Logical Operators and Expressions

Consider an operand 12, for example. Xem xét một toán hạng là 12, chẳng hạn. Bitwise operators will treat this number 12 as 1100. Các toán tử bitwise sẽ coi số 12 này là 1100. Bitwise operators treat the operands as bits rather than numerical values. Các toán tử bitwise xem xét các toán hạng như là các bit thay vì các giá trị số. The numerical values could be of any base: decimal, octal, or hexadecimal. Các giá trị số có thể thuộc bất kỳ hệ số nào: thập phân, bát phân hoặc thập lục phân. The Bitwise operator will convert the operand to its binary representation and accordingly return 1 or a 0. Toán tử bitwise sẽ chuyển đổi toán hạng thành biểu diễn nhị phân của nó và trả về 1 hoặc 0 tương ứng.

Bitwise logical operators are &, | , ^, ~, etc., the details of each of which are summarized in Table 4.3 below.

Operator	Description
Bitwise AND (x & y)	Each bit position returns a one if bits of both operands are ones.
Bitwise OR (x | y)	Each bit position returns a one if bits of either operand are ones.
Bitwise NOT (~ x)	Reverses the bits of its operand.
Bitwise XOR (x ^ y)	Each bit position returns a one if either bits of the operands are ones but not both.

Table 4.3: Bitwise logical operators

Concepts

Bitwise operators treat number types as a 32-bit value, which changes bits according to the operator and then converts them back to the number, when done. For example,

The expression 10 & 15 implies (1010 & 1111) which returns a value 1010 which means 10.

The expression 10 | 15 implies (1010 | 1111) which returns a value 1111 which means 15.

The expression 10 ^ 15 implies (1010 ^ 1111) which returns a value 0101 which means 5.

The expression ~10 implies (~1010) which returns a value -11.

4.5 Mixed Mode Expressions & Type Conversions

A mixed mode expression is one in which the operands of an operator belong to different data types. Một biểu thức hỗn hợp là biểu thức mà các toán hạng của một toán tử thuộc các kiểu dữ liệu khác nhau. These operands are generally converted to the same type. Các toán hạng này thường được chuyển đổi về cùng một kiểu. All operands are converted to the data type of the largest operand. Tất cả các toán hạng được chuyển đổi về kiểu dữ liệu của toán hạng lớn nhất. This is called type promotion. Điều này được gọi là khuyến khích kiểu dữ liệu.
The order of the various data types is: Thứ tự của các kiểu dữ liệu khác nhau là:

`char < int < long < loat < double`

The **automatic type** conversions for evaluating an expression are tabulated below

a. `char` and `short` are converted to `int`, and `float` is converted to `double`

b. If either operand is `double`, the other is converted to `double`, and the result is `double`

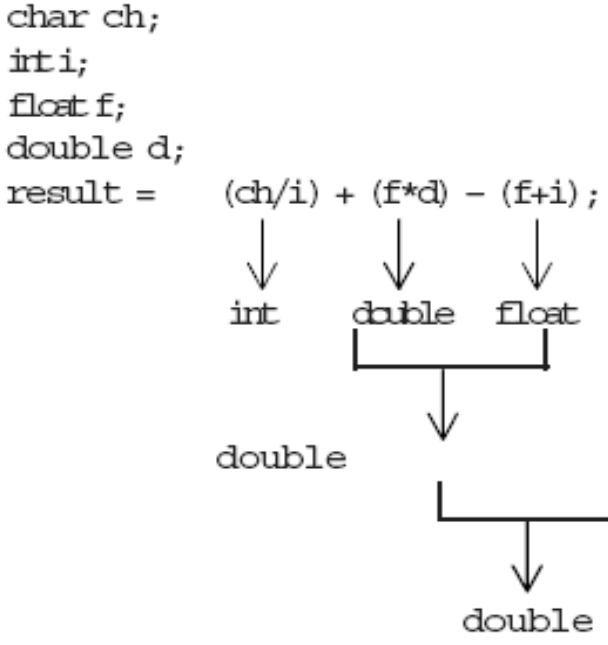
c. If either operand is `long`, the other is converted to `long`, and the result is `double`

d. If either operand is `unsigned`, the other is also converted to unsigned, and the result is also `unsigned`

e. Otherwise all that are left are the operands of type `int`, and the result is `int`

One rule not discussed above is when one operand is `long` and the other is unsigned, and the value of the `unsigned` cannot be represented by `long`. In this case, both operands are converted to `unsigned long`.

Once these conversion rules have been applied, each pair of operands is of the same type and the result of each operation is the same as the type of both operands.



In the above example, first the character **ch** is converted to an integer and `float` **f** is converted to `double`. Then the outcome of **ch/i** is converted to a `double` because **f*d** is `double`. The final result is `double` because, by this time, both operands are `double`.

4.5.1 Casts

It is normally a good practice to change all `integer` constants to `float` if the expression involves arithmetic operations with real values; otherwise some expressions may lose their true value. For example, consider

```
int x,y,z;
x = 10;
y = 100;
z = x/y;
```

In this case **z** will be assigned 0 as integer division takes place and the fractional part (0.10) is truncated.

An expression can be forced to be of a certain type by using a **cast**. The general syntax of cast is:

```
(type) cast
```

where **type** is a valid C data type. For example to make sure that an expression **a/b**, where **a** and **b** are integers, evaluates to a `float`, the following line of code can be written:

Concepts

```
(float) a/b;
```

The cast operator can be used on constants, expressions as well as variables as shown below:

```
(int) 17.487;
(double) (5 * 4 / 8);
(float) (a + 7));
```

In the second example, the cast operator does not serve its purpose because it is performed only after the entire expression within the parentheses is evaluated. The expression `5 * 4 / 8` evaluates to `2` (due to integer truncation), so the resultant `double` value is equal to `2.0`.

Another example:

```
int i = 1, j = 3;
x = i / j; /* x 0.0 */
x = (float) i/(float) j; /* x 0.33 */
```

4.6 Precedence of Operators

Precedence establishes the hierarchy of one set of operators over another when an arithmetic expression has to be evaluated. Quyền ưu tiên xác định thứ bậc của một tập hợp các toán tử so với tập hợp khác khi một biểu thức số học cần được đánh giá. In short, it refers to the order in which C evaluates operators. Tóm lại, nó đề cập đến thứ tự mà C đánh giá các toán tử. The precedence of the arithmetic operators is given in Table 4.4 below. Quyền ưu tiên của các toán tử số học được đưa ra trong Bảng 4.4 dưới đây.

Operator Class	Operators	Associativity
Unary	- ++ – –	Right to left
Binary	^	Left to Right
Binary	* / %	Left to Right
Binary	+ -	Left to Right
Binary	=	Right to Left

Table 4.4: Order of precedence of the arithmetic operators

Operators in the same line in the above table have the same precedence. Các toán tử ở cùng một hàng trong bảng trên có cùng quyền ưu tiên. The evaluation of operators in an arithmetic expression takes place from left to right for operators having equal precedence. Việc đánh giá các toán tử trong một biểu thức số học diễn ra từ trái sang phải đối với các toán tử có cùng quyền ưu tiên. The operators *, /, and % have the same precedence, which is higher than that of + and - (binary). *Các toán tử , / và % có cùng quyền ưu tiên, cao hơn so với quyền ưu tiên của + và - (nhị phân).

The precedence of these operators can be overridden by enclosing the required part of the expression in parentheses (). Quyền ưu tiên của các toán tử này có thể bị ghi đè bằng cách đặt phần cần thiết của biểu thức trong dấu ngoặc đơn (). An expression within parentheses is always evaluated first. Một biểu thức trong dấu ngoặc đơn luôn được đánh giá trước tiên. One set of parentheses may be enclosed within another. Một tập hợp dấu ngoặc đơn có thể được bao gồm trong một tập hợp khác. This is called nesting of parentheses. Điều này được gọi là lồng ghép dấu ngoặc đơn. In such a case, evaluation is done by expanding the innermost parentheses first and then working to the outermost pair of parentheses. Trong trường hợp này, việc đánh giá được thực hiện bằng cách mở rộng các dấu ngoặc đơn ở trong cùng trước và sau đó làm việc ra ngoài cặp dấu ngoặc đơn ở ngoài cùng.

If there are several sets of parentheses in an expression then evaluation takes place from left to right.

Associativity tells how an operator associates with its operands. Tính kết hợp cho biết cách mà một toán tử liên kết với các toán hạng của nó. For example, the unary operator associates with the quantity to its right, and in division, the left operand is divided by the right. Ví dụ, toán tử đơn liên kết với giá trị bên phải của nó, và trong phép chia, toán hạng bên trái được chia cho toán hạng bên phải. The assignment operator (=) associates from right to left. Toán tử gán (=) liên kết từ phải sang trái. Hence, the expression on the right is evaluated first and its value is assigned to the variables on the left. Do đó, biểu thức bên phải được đánh giá trước tiên và giá trị của nó được gán cho các biến bên trái.

Associativity also refers to the order in which C evaluates operators in an expression having same precedence. Such operators can operate either left to right or vice versa as shown in Table 4.5.
For example,

```
a = b =10/2;
```

assigns the value `5` to **`b`** which is then assigned to **`a`**. Hence the associativity is said to be from right to left. Also,

```
-8 * 4 % 2 - 3
```

is evaluated in the following sequence:

Sequence	Operation done	Result
1.	- 8 (unary minus)	negative of 8
2.	- 8 * 4	- 32
3.	- 32 % 2	0
4.	0-3	-3

In the above, the unary minus is evaluated first as it has highest precedence. The evaluation of `*` and `%` takes place from left to right. This is followed by evaluation of the binary `-` operator.

Order of Sub-expressions

Complex expressions could contain smaller expressions within them called sub expressions. C does not specify in what order the sub-expressions are evaluated. An expression like

```
a * b / c + d * c;
```

guarantees that the sub-expression a * b / c and d * c will be evaluated before addition. đảm bảo rằng các biểu thức con a * b / c và d * c sẽ được đánh giá trước phép cộng. Furthermore, the left-to-right rule for multiplication and division guarantees that a will be multiplied by b and then the product will be divided by c. Hơn nữa, quy tắc từ trái sang phải cho phép nhân và chia đảm bảo rằng a sẽ được nhân với b và sau đó tích sẽ được chia cho c. But there is no rule for governing whether a * b / c is evaluated before or after d * c. Nhưng không có quy tắc nào quy định liệu a * b / c sẽ được đánh giá trước hay sau d * c. This option is left open to the designers of the compiler to decide. Tùy chọn này để lại cho các nhà thiết kế trình biên dịch quyết định. The left-to-right rule just applies to a sequence of operators at the same level. Quy tắc từ trái sang phải chỉ áp dụng cho một chuỗi các toán tử ở cùng một cấp độ. That is, it applies to multiplication and division in a * b / c. Có nghĩa là, nó áp dụng cho phép nhân và chia trong a * b / c. But it ends there since the next operator + has a different precedence level. Nhưng nó dừng lại ở đó vì toán tử tiếp theo + có một cấp độ ưu tiên khác.

Concepts

Because of this indeterminacy, expressions whose final value depends upon the order in which the sub-expressions are evaluated should not be used. Consider the following example,

```
a * b + c * b ++ ;
```

One compiler might evaluate the left term first and use the same value of **b** in both the subexpressions. But, a second compiler might evaluate the right terms first and increment **b** before it is used in the left term.

It is preferable that increment or decrement operators should not be used on a variable that appears more than once in an expression.

Precedence between Comparison Operators

Some arithmetic operators, as we saw in the previous section, have some kind of precedence over others. There is no such precedence among comparison operators. They are therefore always evaluated left to right.

Precedence for Logical Operators

The table below presents the order of precedence for logical operators.

Precedence	Operator
1	NOT
2	A N D
3	OR

Table 4.5: Order of precedence for logical operators

When multiple instances of a logical operator are used in a condition, they are evaluated from right to left.

For example, consider the following condition:

False **OR** True **AND NOT** False **AND** True

This condition gets evaluated as shown below:

1. False **OR** True **AND** [**NOT** False] **AND** True **NOT** has the highest precedence.

2. False **OR** True **AND** [True **AND** True]
   Here, **AND** is the operator of the highest precedence and operators of the same precedence are evaluated from right to left.

3. False **OR** [True **AND** True]

4. [False **OR** True]

5. True

Precedence among the Different Types of Operators

When an equation uses more than one type of operator then the order of precedence has to be established with the different types of operators.

The given table presents the precedence among the different types of operators.

Precedence	Type of Operator
1	Arithmetic
2	Comparison
3	Logical

Table 4.6: Order of precedence among different types of operators

Thus, in an equation involving all three types of operators, the arithmetic operators are evaluated first followed by comparison and then logical. The precedence of operators within a particular type has been defined earlier.

Consider the following example:

```
2*3+4/2 > 3 AND 3<5 OR 10<9
```

The evaluation is as shown:

1. `[2*3+4/2] > 3 AND 3<5 OR 10<9`

   First the arithmetic operators are dealt with. The order of precedence for the evaluation arithmetic operators is as in Table 6.

2. `[[2*3]+[4/2]] > 3 AND 3<5 OR 10<9`

3. `[6+2] >3 AND 3<5 OR 10<9`

4. `[8 >3] AND [3<5] OR [10<9]`

   Next to be evaluated are the comparison operators all of which have the same precedence and so

are evaluated from left to right.

5. `True AND True OR False`

   The last to be evaluated are the logical operators. AND takes precedence over OR.

6. `[True AND True] OR False`

7. `True OR False`

8. `True`

The Parentheses (or brackets)

The precedence of operators can be modified by using the parentheses to specify which part of the equation is to be solved first.

- When there are parentheses within parentheses, the innermost parentheses are to be evaluated first.

- When an equation involves multiple sets of parentheses, they are evaluated left to right.

Consider the following example:

```
5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR (2<6 AND 10>11))
```

The solution is:

1. `5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR (True AND False))`

   The inner parenthesis takes precedence over all other operators and the evaluation within this is as per the regular conventions.

2. `5+9*3^2-4 > 10 AND (2+2^4-8/4 > 6 OR False)`

3. `5+9*3^2-4 > 10 AND (2+16-8/4 > 6 OR False)`

   Next the outer parentheses is evaluated Refer to the tables for the precedence of the various operators used.

4. `5+9*3^2-4 > 10 AND (2+16-2 > 6 OR False)`

5. `5+9*3^2-4 > 10 AND (18-2 > 6 OR False)`

6. `5+9*3^2-4 > 10 AND (16 > 6 OR False)`

7. `5+9*3^2-4 > 10 AND (True OR False)`

8. `5+9*3^2-4 > 10 AND True`

9. `5+9*9-4>10 AND True`

   The expression to the left is evaluated as per the conventions.

10. `5+81-4>10 AND True`

11. `86-4>10 AND True`

12. `82>10 AND True`

13. `True AND True`

14. `True`

Concepts

Summary

- C defines four classes of operators: arithmetic, relational, logical, and bitwise.
- All operators in C follow a precedence order.
- Relational operators are used to test the relationship between two variables, or between a variable and a constant.
- Logical operators are symbols that are used to combine or negate expressions containing relational operators.
- Bitwise operators treat the operands as bits rather than numerical value.
- Assignment (=) is considered an operator with right to left associativity.
- Precedence establishes the hierarchy of one set of operators over another when an expression has to be evaluated.

Check Your Progress

1. _____ are the tools that manipulate data.

   A. Operators B. Operands

   C. Expressions D. None of the above

2. An _____ consists of a combination of operators and operands.

   A. Expression B. Functions

   C. Pointers D. None of the above

3. _____ establishes the hierarchy of one set of operators over another when an arithmetic expression has to be evaluated.

   A. Operands B. Precedence

   C. Operator D. None of the above

4. _____ is one in which the operands of an operator belong to different data types.

   A. Single Mode Expression B. Mixed Mode Expression

   C. Precedence D. None of the above

5. An expression can be forced to be of a certain type by using a _____.

   A. Cast B. Precedence

   C. Operator D. None of the above

6. _____ are used to combine or negate expressions containing relational operators.

   A. Logical Operators B. Bitwise Operators

   C. Complex Operators D. None of the above

Concepts

Check Your Progress

7. Bitwise logical operators are __, ___, __ and __ .

   A. % , ^ , * and @ B. &, |, ~ and ^

   C. !,], & and * D. None of the above

8. The precedence of operators can be overridden by enclosing the required part of the expression in _____.

   A. Curly Brackets B. Caret Symbol

   C. Parentheses D. None of the above

Try It Yourself

1. Write a program to accept and add three numbers.

2. For the following values, write a program to evaluate the expression

```
z = a*b+(c/d)-e*f;
```

a=10

b=7

c=15.75

d=4

e=2

f=5.6

3. Write a program to evaluate the area and perimeter of the rectangle.

4. Write a program to evaluate the volume of a cylinder.

5. Write a program to evaluate the net salary of an employee given the following constraints:

Basic salary : $ 12000

DA : 12% of Basic salary

HRA : $150

TA : $120

Others : $450

Tax cuts – a) PF :14% of Basic salary and b) IT: 15% of Basic salary

Net Salary = Basic Salary + DA + HRA + TA + Others – (PF + IT)


“Learn all you can from the
mistakes of others. You won’t have
time to make them all yourself”

Session

5 Operators and Expressions (Lab)

Lab Guide

Objectives

At the end of this session, you will be able to:

- *Use of the arithmetic, comparison, and logical operators*
- *Use type conversions*
- *Understand the precedence among operators*

The steps given in this session are carefully thought and explained in great details so that the understanding of the tool is complete. Follow the steps carefully.

In this section you will write a program to calculate the simple interest on a loan taken.

The formula for calculating the simple interest is **`p * n * r /100`**. Here '**`p`**' stands for principal amount, '**`n`**' stands for number of years and '**`r`**' stands for rate of interest.

The program declares three '**`float`**' variables named **`p`**, **`n`** and **`r`**. Note that the variables are declared in the same line of code using a comma (,) to separate one from the other. Each of these variables is assigned (or given) a value.

Consider the following line of code:

```
printf("\nAmount is : %f", p*n*r/100);
```

In '`printf( )`' above, we have used '`%f`'. '`%f`' is used to display the value of the '`float`' variables mentioned after the comma at the end of the '`printf()`'. In '`printf( )`' the formula to calculate the simple interest is given. That is **`p`**, **`n`** and **`r`** are multiplied and the product is then divided by **`100`**. Thus '`printf()`' displays the amount of simple interest.

Invoke Borland C.

5.1 Calculating Simple Interest

1. **Create a new file.**
2. **Type the following code in the 'Edit window' :**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    float p,n,r;
    clrscr();
    p = 1000;
    n = 2.5;
    r = 10.5;
    printf("\n Amount is : %f", p*n*r/100);
}
```

3. **Save the file with the name `simple.c`**

4. **Compile the file `simple.c`**

5. **Execute the program `simple.c`**

6. **Return to the editor.**

OUTPUT :

```
The Amount is: 262.500000
```

5.2 Using Arithmetic Operators

In this section, you will write a program using the arithmetic operators.

This program declares four 'integer' variables named **a**, **b**, **c** and **d**. Value is assigned to the variables **a**, **b** and **c**.

Consider the following line of code:

```
d = a*b+c/2;
```

a is multiplied by **b**. **c** is divided by **2**. Then the product of **a*b** is added to the quotient (or division) of **c/2**. This value is assigned to the variable **d** using the (=) operator. The expression is evaluated as :

1. `50 * 24 = 1200`

2. `68 / 2 = 34`

3. `1200 + 34 = 1234`

4. `d = 1234`

'`printf( )`' displays the value of the variable **`d`**.

See the last expression:

```
d = a*(b+c+(a-c)*b);
```

Here, the inner parenthesis (or brackets) is having the highest precedence. So **`(a-c)`** is calculated first. After that the outer parentheses expression is calculated in the manner explained. The result of **`(a-c)`** is multiplied with **`b`** because '`*`' has the highest precedence than '`-`' and '`+`' . The expression is evaluated as:

1. **`50*(24+68+(50-68)*24)`**

2. **`50*(24+68+-18*24)`**

3. **`50*(24+68+-432)`**

4. **`50*(92-432)`**

5. **`50*-340`**

6. **`d = -17000`**

Other expressions are evaluated according to the operators used. The result is displayed by the 'printf( )'.

1. Create a new file.

2. Type the following code in the 'Edit Window':

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a,b,c,d;
    clrscr();
    a = 50;
    b = 24;
```

```
    c = 68;
    d = a*b+c/2;
    printf("\n The value after a*b+c/2 is : %d",d);
    d = a%b;
    printf("\n The value after a mod b is : %d",d);
    d = a*b-c;
    printf("\n The value after a*b-c is : %d",d);
    d = a/b+c;
    printf("\n The value after a/b+c is : %d",d);
    d = a+b*c;
    printf("\n The value after a+b*c is : %d",d);
    d = (a+b)*c;
    printf("\n The value after (a+b)*c is : %d",d);
    d = a*(b+c+(a-c)*b);
    printf("\n The value after a*(b+c+(a-c)*b) is : %d",d);
}
```

3. **Save the file with the name `arith.c`**
4. **Compile the file `arith.c`**
5. **Execute the program `arith.c`**
6. **Return to the editor.**

OUTPUT :

```
The value after a*b+c/2 is : 1234
The value after a mod b is : 2
The value after a*b-c is : 1132
The value after a/b+c is : 70
The value after a+b*c is : 1682
The value after (a+b)*c is : 5032
The value after a*(b+c+(a-c)+b) is : -17000
```

5.3 Using Comparison and Logical operators

In this section you will write a program using the comparison and logical operators.

Three integer variables named **`a`**, **`b`** and **`c`** are declared in this program. Values are assigned to the

variables. Let **a = 5**, **b = 6** & **c = 7**.

Consider the following lines of code:

1. `a + b >= c;`

   First **a+b** will be calculated (arithmetic operators have higher precedence than comparison operators) i.e. , **11**. Next the value **11** is compared with **c**. The result is **1** (**true**) because **11 > 7**.

2. Consider another expression:

   `a > 10 && b < 5 ;`

Here the first calculation will be `a > 10` and `b < 5`, because comparison operators (> <) have more precedence than logical AND operator (`&&`). The equation will be evaluated as:

1. 5 > 10 && 6 < 5

2. **FALSE && FALSE**

3. **FALSE** i.e., **0**

1. Create a new file.

2. Type the following code in the 'Edit Window':

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a = 5, b = 6, c = 7;
    printf ("int a = 5, b = 6, c = 7;\n");
    printf("The value of a > b is \t%i\n\n", a > b);
    printf("The value of b < c is \t%i\n\n", b < c);
    printf("The value of a + b >= c is \t%i\n\n", a + b >= c);
    printf("The value of a - b <= b - c is\t%i\n\n", a-b<=b-c);
    printf("The value of b-a =b - c is\t%i\n\n", b - a == b - c);
    printf("The value of a*b!= c * c is\t%i\n\n", a * b < c * c);
    printf("Result of a>10 && b <5 = %d\n\n",a>10&&b<5);
    printf("Result of a>100 || b<50=%d\n\n",a>100 || b<50);
}
```

3. **Save the file with the name `compare.c`**

4. **Compile the file `compare.c`**

5. **Execute the program `compare.c`**

6. **Return to the editor.**

OUTPUT :

```
int a = 5, b = 6, c = 7;
The value of a > b is 0
The value of b < c is 1
The value of a + b >= c is 1
The value of a - b <= b - c is 1
The value of b - a == b - c is 0
The value of a * b != c * c is 1
Result of a > 10 && b <5 = 0
Result of a>100 || b<50 = 1
```

5.4 Using Type-conversion

In this section you will write a program to understand type conversion.

In the first expression, all are entirely '`int`' context, **`40 / 17 * 13 / 3`** would evaluate to **`8 (40 / 17`** rounds to **`2, 2 * 13 = 26`**, **`26 / 3`** rounds to **`8`**)

The second expression:

1. to evaluate **`40 / 17 * 13 / 3.0`**

2. **`40 / 17`** again rounds to **`2`**

3. 2 * 13 = 26

4. but **`3.0`** forces the final division into a double context, thus **`26.0 / 3.0 = 8.666667`**

In the third expression, if we move the decimal point to **`13 (40 / 17 * 13.0 / 3)`**, the result will still be **`8.666667`** because:

1. **`40 / 17`** rounds to **`2`**

2. the `13.0` forces the multiplication to a double but `2.0 * 13.0` still equals `26.0`

3. and `26.0` still forces the final division into a double context, so `26.0 / 3.0 = 8.666667`

In the last expression, if we move the decimal point to `17 (40 / 17.0 * 13 / 3)`, the result now becomes `10.196078` because:

1. `17.0` forces the initial division into a double context and `40.0 / 17.0 = 2.352941`

2. `2.352941 * 13.0 = 30.588233`

3. and `30.588233 / 3.0 = 10.196078`

1. Create a new file.

2. Type the following code in the 'Edit Window' :

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("40/17*13/3 = %d",40/17*13/3);
    printf("\n\n40/17*13/3.0 = %lf",40/17*13/3.0);
    printf("\n\n40/17*13.0/3 = %lf",40/17*13.0/3);
    printf("\n\n40/17.0*13/3 = %lf",40/17.0*13/3);
}
```

3. Save the file with the name `type.c`

4. Compile the file `type.c`

5. Execute the program `type.c`

6. Return to the editor.

OUTPUT :

```
40/17*13/3 = 8
40/17*13/3.0 = 8.666667
40/17*13.0/3 = 8.666667
40/17.0*13/3 = 10.196078
```

Lab Guide

5.5 Precedence of operators

In this section you will write a program to see the precedence among operators.

The following expression will be evaluated as follows:

```
(4-2*9/6<=3 && (10*2/4-3>3|| (1<5 && 8>10)))
```

Follow the rules that we have studied in the session “Operators and Expressions”

(Note that the expressions shown as bold below are evaluated first)

1. (4-2*9/6<=3 && (10*2/4-3>3|| **(1<5 && 8>10)**))
2. (4-2*9/6<=3 && (10*2/4-3>3|| **(1 && 0)**))
3. (4-2*9/6<=3 && **(10*2/4-3>3**|| **0)**)
4. (4-2*9/6<=3 && **(20/4-3>3**|| **0)**)
5. (4-2*9/6<=3 && **(5-3>3**|| **0)**)
6. (4-2*9/6<=3 && **(2>3**|| **0)**)
7. (4-2*9/6<=3 && **(0**|| **0)**)
8. (4-**2*9**/6<=3 && 0)
9. (4-**18/6**<=3 && 0)
10. (**4-3**<=3 && 0)
11. (**1<=3** && 0)
12. (**1 && 0**)
13. **0** (**False**)

1. Create a new file.

2. Type the following code in the ‘Edit Window’ :

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Result=%d",(4-2*9/6<=3&&(10*2/4-3>3||(1<5&&8>10))));
}
```

3. **Save the file with the name `precede.c`**

4. **Compile the file `precede.c`**

5. **Execute the program `precede.c`**

6. **Return to the editor.**

OUTPUT :

```
Result = 0
```

Part II: For the next 30 Minutes:

1. Solve the following expression:

```
10 * 3 ^ 6 * 6 + 5 – 2 AND (2 * 2 + 6 /3 > 1 OR 2 >8)
```

   To do this :

   Type the above expression using `printf()` statement . AND can be replaced by `&&` (ampersand symbol) and OR can be replaced by `||` (double pipe symbol)

2. Assume all the variables are of type `int`. Find the values of each of the following variables:

```
a.  x = (2+3) * 6;

b.  x = (12 + 6) /2 *3;

c.  y = x = (2 + 3) / 4;

d.  y = 3 + 2 *(x = 7/2);

e.  x = (int)3.8 + 3.3;

f.  x = (2 + 3) * 10.5;

g.  x = 3/5 * 22.0;

h.  x = 22.0 * 3/5;
```

Try It Yourself

1. What is the assigned (left-hand side) value in each case?

```
int s, m=3, n=5, r, t;

float x=3.0, y;

t = n/m;

r = n%m;

y = n/m;

t = x*y-m/2;

x = x*2.0;

s = (m+n)/r;

y = --n;
```

2. Write a program which will take the input as a floating (real) number. This number represents the centimeters. Print out the equivalent number of feet(floating, 1 decimal) and inches (floating, 1 decimal), with feet and the inches given to an accuracy of one decimal place.

   Assume 2.54 centimeters per inch, and 12 inches per foot.

   If the input value is 333.3, the output format should be:

   333.3 centimeters is 10.9 feet

   333.33 centimeters is 131.2 inches

3. Find the value of iResult for the following assignment statements:

```
int iResult, a = 10, b = 8, c = 6, d = 5, e =2;
```

Try It Yourself

```
iResult = a - b - c - d;

iResult = a - b + c - d;

iResult = a + b / c / d;

iResult = a + b / c * d;

iResult = a / b * c * d;

iResult = a % b / c * d;

iResult = a % b % c % d;

iResult = a - (b - c) - d;

iResult = (a - (b - c)) - d;

iResult = a - ((b - c) - d);

iResult = a % (b % c) * d * e;

iResult = a + (b - c) * d - e;

iResult = (a + b) * c + d * e;

iResult = (a + b) * (c / d) % e;
```

Session

6 Input and Output in 'C'

Concepts

Objectives

At the end of this session, you will be able to:

- *To understand formatted I/O functions scanf() and printf()*
- *To use character I/O functions getchar() and putchar()*

Introduction

In any programming language, assigning values to variables and printing them after processing can be done in two ways,

- Through standard Input/ Output (I/O) media
- Through files

This session covers basic input and output. Usually input and output (I/O), form an important part of any program. To do anything useful, your program needs to be able to accept input data and report back your results. In C, the standard library provides routines for input and output. The standard library has functions for I/O that handle input, output, and character and string manipulation. In this lesson, all the input functions described read from standard input and all the output functions described write to standard output. Standard input is usually the keyboard. Standard output is usually the monitor (also called the console). Input and Output can be rerouted from or to files instead of the standard devices. The files may be on a disk or on any other storage medium. The output may be sent to the printer also.

6.1 The Header File <stdio.h>

In the examples, given so far you would have seen the following line,

```
#include <stdio.h>
```

This is a preprocessor command. In standard C, the `#` should be in the first column. **`stdio.h`** is a file and is called the header file. It contains the macros for many of the input/output functions used in C. The `printf()`, `scanf()`, `putchar()` and `getchar()` functions are designed in such a way that they require the macros in **`stdio.h`** for proper execution.

Concepts

6.2 Input and Output in C

The standard library in C provides two functions that perform formatted input and output. They are:

- `printf()` – for formatted output
- `scanf()` – for formatted input

These functions are called formatted functions as they can read and print data in various prespecified formats which are under the user's control. Format specifiers specify the format in which the values of the variables are to be input and printed.

6.2.1 printf()

You are already familiar with this function, since it has been used in the earlier sessions. We shall have a detailed look at the function here. The function `printf()` is used to display data on the standard output – console. The general format of the function is:

```
printf( "control string", argument list);
```

The argument list consists of constants, variables, expressions or functions separated by commas. There must be one format command in the control string for each argument in the list. The format commands must match the argument list in number, type and order. The control string must always be enclosed within double quotes(" "), which are its delimiters. The control string consists of one or more of three types of items which are explained below:

- **Text characters** – This consists of printable characters that are to be printed as they are. Spaces are often used to separate output fields.
- **Format Commands** define the way the data items in the argument list are to be displayed. A format command begins with a % sign and is followed by a format code appropriate for the data item. % is used by the `printf()` function to identify conversion specifications. The format commands and the data items are matched in order and typed from left to right. One format code is required for every data item that has to be printed.
- **Nonprinting Characters** – This includes tabs, blanks and new lines.

Each format command consists of one or more format codes. A format code consists of a % and a type specifier. Table 6.1 lists the various format codes supported by the `printf()` statement:

Format	printf()	scanf()
Single Character	%c	%c
String	%s	%s

Format	printf()	scanf()
Signed decimal integer	%d	%d
Floating point (decimal notation)	%f	%f or %e
Floating point (decimal notation)	%lf	%lf
Floating point (exponential notation)	%e	%f or %e
Floating point (%f or %e , whichever is shorter)	%g	
Unsigned decimal integer	%u	%u
Unsigned hexadecimal integer (uses "ABCDEF")	%x	%x
Unsigned octal integer	%o	%o

Table 6.1: printf() Format Codes

In the above table **`c`**, **`d`**, **`f`**, **`lf`**, **`e`**, **`g`**, **`u`**, **`s`**, **`o`** and **`x`** are the type specifiers.

The printing conventions of the various format codes are summarized in Table 6.2:

Format Code	Printing Conventions
%d	The number of digits in the integer.
%f	The integer part of the number will be printed as such. The decimal part will consist of 6 digits. If the decimal part of the number is smaller than 6, it will be padded with trailing zeroes to the right, else it will be rounded at the right.
%e	One digit to the left of the decimal point and 6 places to the right , as in %f above.

Table 6.2: Printing Conventions

Since %, \ and " have special uses in the control string , if they have to be inserted as part of a text string and printed on the console, they must be used as seen in Table 6.3 :

\\	to print \ character
\	" to print " character
% %	to print % character

Table 6.3: Control String Special Characters

Concepts

No	Statements	Control String	What the control string contains	Argument List	Explanation of the argument list	Screen Display
1.	`printf("%d",300);`	`%d`	Consists of format command only	300	Constant	300
2.	`printf("%d",10+5);`	`%d`	Consists of format command only	10 + 5	Expression	15
3.	`printf("Good Morning Mr. Lee.");`	`Good Morning Mr. Lee.`	Consists of text characters only	Nil	Nil	Good Morning Mr. Lee.
4.	`int count = 100;` `printf("%d",count);`	`%d`	Consists of format command only	Count	Variable	100
5.	`printf("\nhello");`	`\nhello`	Consists of nonprinting character & text characters	Nil	Nil	hello on a new line

6.	`#define str "`
	`Good Apple "`
	`……..`
	`printf("%s",str);`
		`%s`	Consists of format command only	Str	Symbolic constant	Good Apple

7.	`……`
	`int count,stud_num;`
	`count=0;`
	`stud_nim=100;`
	`printf("%d %d\n",`
	`count, stud_num);`
		`%d %d`	Consists of format command and escape sequence	count, stud_num	Two variables	0 , 100

Table 6.4: Control Strings and Format Codes

Example 1:

```
/* This is a simple program which demonstrates how a string can be printed
from within a format command and also as an argument. This program also
displays a single character, integer, and float. */

#include <stdio.h>
void main()
{
    int a = 10;
```

```
    float b = 24.67892345;
    char ch = 'A';
    printf("Integer data = %d", a);
    printf("Float Data = %f",b);
    printf("Character = %c",ch);
    printf("This prints the string");
    printf("%s","This also prints a string");
}
```

A sample run is shown below:

```
Integer data = 10
Float Data = 24.678923
Character = A
This prints the string
This also prints a string
```

➢ **Modifiers for Format Commands in printf()**

The format commands may have modifiers, to suitably modify the basic conversion specifications. The following are valid modifiers acceptable in the `printf()` statement. If more than one modifier is used, then they must be in the same order as given below.

'-' Modifier

The data item will be left-justified within its field; the item will be printed beginning from the leftmost position of its field.

Field Width Modifier

They can be used with type `float`, `double` or `char` array (`string`). The field width modifier, which is an integer, defines the minimum field width for the data item. Data items for smaller width will be output right-justified within the field. Larger data items will be printed by using as many extra positions as required. e.g. **`%10f`** is the format command for a type `float` data item with minimum field width 10.

Precision Modifier

This modifier can be used with type `float`, `double` or `char` array (`string`). The modifier is written as **`.m`** where **`m`** is an integer. If used with data type `float` or `double`, the digit string indicates the maximum number of digits to be printed to the right of the decimal. When used with a string, it indicates the maximum number of characters to be printed.

If the fractional part of a type `float` or `double` data item exceeds the precision modifier, then the number will be rounded. If a string length exceeds the specified length, then the string will be truncated (cut of at the end). Padding with zeroes occurs for numbers when the actual number of digits in a data item is less than that specified by the modifier. Similarly blanks are padded for strings. e.g. **`%10.3f`** is the format command for a type float data item, with minimum field width 10 and 3 places after the decimal.

'0' Modifier

The default padding in a field is done with spaces. If the user wishes to pad a field with zeroes, this modifier must be used.

'1' Modifier

This modifier can be used to display integers as long int or a double precision argument. The corresponding format code is `%ld`.

'h' Modifier

This modifier is used to display short integers. The corresponding format code is `%hd`.

'*' Modifier

This modifier is used if the user does not want to specify the field width in advance, but wants the program to specify it. But along with this modifier an argument is required which tells what the field width should be.

Let us now see how these modifiers work. First we see their effect when used with integer data items.

Example 2:

```
/* This program demonstrates the use of Modifiers in printf() */
#include <stdio.h>
void main()
{
    printf("The number 555 in various forms:\n");
    printf("Without any modifier: \n");
    printf("[%d]\n",555);
    printf("With - modifier :\n");
    printf("[%-d]\n",555);
```

```
    printf("With digit string 10 as modifier :\n");
    printf("[%10d]\n",555);
    printf("With 0 as modifier : \n");
    printf("[%0d]\n",555);
    printf("With 0 and digit string 10 as modifiers :\n");
    printf("[%010d]\n",555);
    printf("With -, 0 and digit string 10 as modifiers: \n");
    printf("[%-010d]\n",555);
}
```

A sample run is shown below :

```
The number 555 in various forms:
Without any modifier:
[555]
With - modifier :
[555]
With digit string 10 as modifier :
[555]
With 0 as modifier :
[555]
With 0 and digit string 10 as modifiers :
[0000000555]
With -, 0 and digit string 10 as modifiers:
[555]
```

We have used `[` and `]` to show where the field begins and where it ends. When we use `%d` with no modifiers, we see that it uses a field with the same width as the integer. When using `%10,d` we see that it uses a field 10 spaces wide and the number is right-justified, by default. If we use the – modifier, the number is left-justified in the same field. If we use the 0 modifier, we see that the number is padded with 0's instead of blanks.

Now let us see how modifiers can be used with floating-point numbers.

Example 3:

```
/* This program demonstrates the use of Modifiers in printf() */
#include <stdio.h>
void main()
{
```

Concepts

```
    printf("The number 555.55 in various forms:\n");
    printf("In float form without modifiers :\n");
     printf("[%f]\n",555.55); printf("In exponential form without any
modifier: \n");
    printf("[%e]\n",555.55);
    printf("In float form with - modifier:\n");
    printf("[%-f]\n",555.55);
    printf("In float form with digit string 10.3 as
modifier\n");
    printf("[%10.3f]\n",555.55);
    printf("In float form with 0 as modifier: \n");
    printf("[%0f]\n",555.55);
    printf("In float form with 0 and digit string 10.3 ");
    printf("as modifiers:\n");
    printf("[%010.3f]\n",555.55);
    printf("In float form with -, 0 ");
    printf("and digit string 10.3 as modifiers: \n");
    printf("[%-010.3f]\n",555.55);
    printf("In exponential form with 0 ");
    printf("and digit string 10.3 as modifiers:\n");
    printf("[%010.3e]\n",555.55);
    printf("In exponential form with -, 0 ");
    printf("and digit string 10.3 as modifiers : \n");
    printf("[%-010.3e]\n\n",555.55);
}
```

A sample output is shown below :

```
The number 555.55 in various forms:
In float form without modifiers:
[555.550000]
In exponential form without any modifier:
[5.555500e+02]
In float form with - modifier:
[555.550000]
In float form with digit string 10.3 as modifier
[555.550]
```

```
In float form with 0 as modifier:
[555.550000]
In float form with 0 and digit string 10.3 as modifiers:
[000555.550]
In float form with -, 0 and digit string 10.3 as modifiers:
[555.550]
In exponential form with 0 and digit string 10.3 as modifiers:
[05.555e+02]
In exponential form with -,0 and digit string 10.3 as modifiers :
[5.555e+02]
```

In the default version of `%f`, we can see that there are six decimal digits and the default specification of `%e` is one digit to the left of the decimal point and six digits to the right of the decimal point. Notice how in the last two examples, the number of digits to the right of the decimal point being 3, causes the output to be rounded off.

Now let us see how modifiers can be used with strings. Notice how the field is expanded to contain the entire string. Also note how the precision specification **.4** limits the number of characters to be printed.

Example 4:

```
/* Program to show the use of modifiers with strings */
#include <stdio.h>
void main()
{
    printf("A string in various forms :\n");
    printf("Without any format command :\n");
    printf("Good day Mr. Lee. \n");
    printf("With format command but without any modifier:\n");
    printf("[%s]\n","Good day Mr. Lee.");
    printf("With digit string 4 as modifier :\n");
    printf("[%4s]\n","Good day Mr. Lee.");
    printf("With digit string 19 as modifier: \n");
    printf("[%19s]\n","Good day Mr. Lee.");
    printf("With digit string 23 as modifier: \n");
    printf("[%23s]\n","Good day Mr. Lee.");
    printf("With digit string 25.4 as modifier: \n");
    printf("[%25.4s]\n","Good day Mr.Lee.");
```

```
    printf("With - and digit string 25.4 as modifiers :\n");
    printf("[%-25.4s]\n","Good day Mr.shroff.");
}
```

A sample output is shown below:

```
A string in various forms :
Without any format command :
Good day Mr. Lee.
With format command but without any modifier:
[Good day Mr. Lee.]
With digit string 4 as modifier :
[Good day Mr. Lee.]
With digit string 19 as modifier:
[Good day Mr. Lee.]
With digit string 23 as modifier:
[ Good day Mr. Lee.]
With digit string 25.4 as modifier:
[ Good]
With - and digit string 25.4 as modifiers :
[Good ]
```

The characters we type at the keyboard are not stored as characters. Instead they are stored as numbers in the **ASCII (American Standard Code for Information Interchange)** code format. The values that a variable holds are interpreted as a character or a numeric depending on the type of the variable that holds it . The following example demonstrates this:

Example 5:

```
#include <stdio.h>
void main()
{
    int a = 80;
    char b= 'C';
    printf("\nThis is the number stored in 'a' %d",a);
    printf("\nThis is a character interpreted from 'a' %c",a);
    printf("\nThis is also a character stored in 'b' %c",b);
    printf("\nHey!the character of 'b' is printed as a number!%d",b);
}
```

A sample output is shown below:

```
This is the number stored in 'a' 80
This is a character interpreted from 'a' P
This is also a character stored in 'b' C
Hey! the character of 'b' is printed as a number !67
```

This result demonstrates the use of format specifications and the interpretation of ASCII codes. Though the variables **a** and **b** have been declared as `int` and `char` variables, they have been printed as character and number using different format specifiers. This feature of C gives flexibility in the manipulation of data.

How can you use the `printf()` statement to print a string which extends beyond a line of 80 characters? You must terminate each line by a \ symbol as shown in the example below:

Example 6

```
/* Program demonstrates how to print a long string */
#include <stdio.h>
void main()
{
    printf("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
}
```

A sample output is shown below:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

In the above example, the string in the `printf()` statement is 252 characters long. Since a line of text contains 80 characters, the string extends beyond three lines in the output shown above.

6.2.2 scanf()

The `scanf()` function is used to accept data. The general format of the `scanf()` function is as given below.

```
scanf("control string", argument list);
```

The format used in the `printf()` statement are used with the same syntax in the `scanf()` statements too.

The format commands, modifiers and argument list discussed for `printf()` is valid for `scanf()` also, subject to the following differences:

- **Differences in argument list of between printf() and scanf()**

  `printf()` uses variable names, constants, symbolic constants and expressions, but `scanf()` uses pointers to variables. A pointer to a variable is a data item which contains the address of the location where the variable is stored in memory. Pointers will be discussed in detail later. When using `scanf()` follow these rules, for the argument list:

  - If you wish to read in the value of a variable of basic data type, type the variable name with & symbol before it.
  - When reading the value of a variable of derived data type, do not use & before the variable name.

- **Differences in the format commands of the printf() and scanf()**

  a. There is no `%g` option.

  b. The `%f` and `%e` format codes are in effect the same. Both accept an optional sign, a string of digits with or without a decimal point, and an optional exponent field.

How scanf() works?

`scanf()` uses nonprinting characters like blank, tab, new line to decide when an input field ends and input field begins. It matches the format commands with the fields in the argument list in the same order in which they are specified, skipping over the white space characters in between. Therefore the input can be spread over more than one line, as long as we have at least one tab, space or new line between each input field. It skips white spaces and line boundaries to obtain the data.

Example 7:

The following program demonstrates the use of the `scanf()` function.

```
#include <stdio.h>
void main()
{
```

```
   int a;
   float d;
   char ch, name[40];
   printf("Please enter the data\n");
   scanf("%d %f %c %s", &a, &d, &ch, name);
   printf("\n The values accepted are : %d, %f, %c, %s", a, d, ch, name);
}
```

A sample output is shown below:

```
Please enter the data
12 67.9 F MARK
The values accepted are :12, 67.900002, F, MARK
```

The input could have been

```
12 67.9

F MARK
```

or even

```
12

67.9

F

MARK
```

for it to be properly accepted into **a**, **d**, **ch**, and **name**.

Consider another example:

Example 8:

```
#include <stdio.h>
void main()
{
    int i;
```

Concepts

```
    float x;
    char c;
    ……….
    scanf("%3d %5f %c",&i,&x, &c);
}
```

If the data items are entered as

```
21 10.345 F
```

When the program is executed, then 21 will be assigned to i , 10.34 will be assigned to **x** and the character 5 will be assigned to **c**. The remaining character **F** will be ignored.

If you specify a field width in `scanf()`, say `%10s`, then `scanf()` collects upto 10 characters or to the first white space character, whichever comes first. This is true for type `int`, `float` and `double` as well.

The example below demonstrates the use of the `scanf()` function to enter a string consisting of uppercase letters and blank spaces. The string will be of undetermined length, but it will be limited to 79 characters (actually, 80 characters including the null character that is added at the end).

Example 9:

```
#include <stdio.h>
void main()
{
    char line[80]; /* line[80] is an array which stores 80 characters */
    ……….
    scanf("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]",line);
    ……….
}
```

The format code `%[]` means that characters defined within `[]` can be accepted as valid string characters. If the string **BEIJING CITY** is entered from the standard input device when the program is executed, the entire string will be assigned to the array line since the string is composed entirely of uppercase letters and blank spaces. If the string was written as Beijing City however, then only the single letter **B** would be assigned to line, since the first lowercase letter (in this case, **e**) would be interpreted as the first character beyond the string.

To accept any character up to a new line character, we use the format code **%[^\n]**, which implies that the string will accept any character except `\n`, which is a new line. The caret (^) implies that all characters except those following the caret will be accepted as valid input characters.

Example 10:

```
#include <stdio.h>
void main()
{
    char line[80];
    ……
    scanf("%[^\n]",line);
    ……
}
```

When the `scanf()` function is executed, a string of undetermined length (but not more than 79 characters) will be entered from the standard input device and assigned to line. There will be no restriction on the characters that compose the string, except that they all fit on one line. For example, the string

```
All's well that ends well !
```

could be entered from the keyboard and assigned to line.

The `*` modifier works differently in `scanf()`. The asterisk is used to indicate that a field is to be ignored or skipped.

For example consider the program:

Example 11:

```
#include <stdio.h>
void main()
{
    char item[20];
    int partno;
    float cost;
    ……
    scanf("%s %*d %f",item, &partno, &cost);
    ……
}
```

If the corresponding data items are

```
battery 12345 0.05
```

then **`battery`** will be assigned to **`item`** and 0.05 will be assigned to cost but 12345 will not be assigned

Concepts

to **partno** because of the assignment suppression character asterisk(*).

Any other character in `scanf()`, which is not part of the format code within the control string, must be typed in input in exactly the same way otherwise it causes errors. This feature is used to accept comma(,) delimited input.

For example consider the data stream

```
10, 15, 17
```

and the input command

```
scanf("%d, %f, %c", &intgr, &flt, &ch);
```

Note that the commas in the conversion string, match the commas in the input stream and hence will serve as delimiters.

White space characters in the control string are normally ignored except that it causes problems with **%c** format code. If we use **%c** specifier, then a space is considered a valid character.

Consider the code segment:

```
int x, y;
char ch;
scanf("%2d %c %d",&x, &ch, &y);
printf("%d %d %d\n",x, ch, y);
```

with the input

```
14 c 5
```

14 will be assigned to **x**, the character **ch** has the space (decimal 99) as its value, and **y** is assigned the value of character **c** which is decimal 99.

Consider the following code:

```
#include <stdio.h>
void main()
{
    char c1, c2, c3;
    ……
```

```
    scanf("%c%c%c",&c1, &c2, &c3);
    ……….
}
```

If the input data is

```
a b c
```

(with blank spaces between the letters), then the following assignments would result

```
c1 = a, c2 = <blank space>, c3 = b
```

Here we can see **c2** contains a blank space because the input contains white space character. To skip over such white space characters and read the next non-white space character, the conversion group **%1s** should be used.

```
scanf("%c%1s%1s",&c1, &c2, &c3);
```

Then the same input of data would result in the following assignments

```
c1 = a, c2 =b, c3 = c
```

as intended.

6.3 Buffered I/O

C language as such does not define input and output operations by itself. All I/O operations are performed by the functions available in the C function library. C function library contains one distinct system of routine that handles these operations. That is:

Buffered I/O – used to read and write ASCII characters

A buffer is a temporary storage area, either in the memory or on the controller card for the device. In buffered I/O, characters typed at the keyboard are collected until the user presses the return or the enter key, when the characters are made available to the program, as a block.

Buffered I/O can be further subdivided into:

- Console I/O
- Buffered File I/O

Concepts

Console I/O refers to operations that occur at the keyboard and the screen of your computer. Buffered File I/O refers to operations that are performed to read and write data onto a file. We will discuss the Console I/O.

In C, console is considered as a stream device. The Console I/O functions direct their operations to the standard input and output of the system.

The simplest Console I/O functions are:

- `getchar()` – which reads one (and only one) character from the keyboard.
- `putchar()` – which outputs a single character on the screen.

6.3.1 getchar()

The function `getchar()` is used to read input data, a character at a time from the keyboard. In most implementations of C, `getchar()` buffers characters until the user types a carriage return. Therefore it waits until the return key is pressed. The `getchar()` function has no argument, but the parentheses must still be present. It simply fetches the next character and makes it available to the program. We say that the function returns a value, which is a character.

The following program demonstrates the use of the `getchar()` function.

Example 12:

```
/* Program to demonstrate the use of getchar() */
#include <stdio.h>
void main()
{
    char letter;
    printf("\nPlease enter any character : ");
    letter = getchar();
    printf("\nThe character entered by you is %c . ", letter);
}
```

A sample output is shown below:

```
Please enter any character: S
The character entered by you is S .
```

In the above program, **`letter`** is a variable declared to be of type `char` so as to accept character input. A message

```
Please enter any character:
```

will appear on the screen. You enter a character, say **s**, through the keyboard, and press carriage return. The function **`getchar()`** fetches the character entered by you and assigns the same to the variable letter. Later it is displayed on the screen with the message.

```
The character entered by you is S .
```

6.3.2 putchar()

`putchar()` is the character output function in C, which displays a character on the screen at the cursor position. This function requires an argument. The argument of the `putchar()` function can be any one of the following:

- A single character constant
- An escape sequence
- A character variable

If the argument is a constant, it must be enclosed in single quotes. Table 6.5 demonstrates some of the options available in `putchar()` and their effect.

Argument	Function	Effect
character variable	`putchar(c)`	Displays the contents of character variable c
character constant	`putchar('A')`	Displays the letter A
numeric constant	`putchar('5')`	Displays the digit 5
escape sequence	`putchar('\t')`	Inserts a tab space character at the cursor position
escape sequence	`putchar('\n')`	Inserts a carriage return at the cursor position

Table 6.5: putchar() options and their effects

The following program demonstrates the working of `putchar()`:

Example 13:

```
/* This program demonstrates the use of constants and escape
sequences in putchar() */
#include <stdio.h>
void main()
{
```

Concepts

```
    putchar('H'); putchar('\n');
    putchar('\t');
    putchar('E'); putchar('\n');
    putchar('\t'); putchar('\t');
    putchar('L'); putchar('\n');
    putchar('\t'); putchar('\t'); putchar('\t');
    putchar('L'); putchar('\n');
    putchar('\t'); putchar('\t'); putchar('\t');
    putchar('\t');
    putchar('O');
}
```

A sample output is shown below:

```
H
   E
      L
          L
             O
```

The difference between `getchar()` and `putchar()` is that `putchar()` requires an argument, while `getchar()` the earlier does not.

Example 14:

```
/* Program demonstrates getchar() and putchar() */
#include <stdio.h>
void main()
{
    char letter;
    printf("You can enter a character now: ");
    letter = getchar();
    putchar(letter);
}
```

A sample output is shown below:

```
You can enter a character now: F

F
```

Summary

- In C, I/O is performed using functions. Any program in C has access to three standard files. They are standard input file (called stdin), standard output file (called stdout) and the standard error (called stderr). Normally the standard input file is the keyboard, the standard output file is the screen and the standard error file is also the screen.
- The header file <stdio.h> contains the macros for many of the input / output functions used in C.
- Console I/O refers to operations that occur at the keyboard and the screen of your computer. It consists of formatted and unformatted functions.
- The formatted I/O functions are printf() and scanf().
- The unformatted functions are getchar() and putchar().
- The scanf() function is used to take the formatted input data, whereas the printf() function is used to print data in the specified format.
- The control string of printf() and scanf() must always be present inside " and ". The string consists of a set of format commands. Each format command consists of a %, an optional set of modifiers and a type specifier.
- The major difference between printf() and scanf() is that the scanf() function uses addresses of the variables rather than the variable names themselves.
- The getchar() function reads a character from the keyboard.
- The putchar(ch) function sends the character ch to the screen.
- The difference between getchar() and putchar() is that putchar() takes an argument while getchar() does not.

Concepts

Check Your Progress

1. The formatted I/O functions are _____ and _____.

   A. printf() and scanf() B. getchar() and putchar()

   C. puts() and gets() D. None of the above

2. scanf() uses _____ to variables rather than variable names.

   A. functions B. pointers

   C. arrays D. None of the above

3. _____ specify the form by which the values of the variables are to be input and printed.

   A. Text B. format specifier

   C. argument D. None of the above

4. ___ is used by the printf() function to identify conversion specifications.

   A. % B. &

   C. * D. None of the above

5. getchar() is a function without any arguments **[T/F]**

6. _____ is a temporary storage area in memory.

   A. ROM B. Register

   C. Buffer D. None of the above

7. Escape sequence can be placed outside the control string in printf(). **[T/F]**

Try It Yourself

1. A. Use the `printf()` statement and do the following :

   a. Print out the value of the integer variable **`sum`**

   b. Print out the text string "**`Welcome`**", followed by a new line.

   c. Print out the character variable **`letter`**

   d. Print out the `float` variable **`discount`**

   e. Print out the `float` variable **`dump`** using two decimal places

   B. Use the `scanf()` statement and do the following:

   a. To read a decimal value from the keyboard, into the integer variable **`sum`**

   b. To read a `float` variable into the variable **`discount_rate`**

2. Write a program which prints the ASCII values of the characters '**`A`**' and '**`b`**'.

3. Consider the following program:

```
#include <stdio.h>
void main()
{
    int breadth;
    float length, height;
    scanf("%d%f%6.2f",breadth,&length,height);
    printf("%d %f %e",&breadth, length, height);
}
```

Correct the errors in the above program.

Concepts

Try It Yourself

4. Write a program which takes **name**, **basic** , **daper** (ie, percentage of D.A), **bonper** (ie, percentage bonus) and **loandet** (loan amount to be debited) for an employee. Calculate the salary using the following relation:

```
salary = basic + basic * daper /100 +bonper * basic/100 - loandet
```

Data is:

name	basic	daper	bonper	loandet
M A R K	2500	55	33.33	250.00

Calculate salary and then print the result under the following headings.

(Salary to be printed to the nearest dollar.)

Name **Basic** **Salary**

5. Write a program that asks for your first name and last name, and then prints the names in the format last name, first name.