

5 Operators and Expressions (Lab)

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*

In this section you will write a program to calculate the simple interest on a loan taken (Trong phần này bạn sẽ viết một chương trình để tính lãi suất đơn của một khoản vay).

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' :

Session 5

Operators and Expressions (Lab)

```
#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

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. $68 / 2 = 34$

Session 5

Operators and Expressions (Lab)

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;
```

Session 5

Operators and Expressions (Lab)

Lab Guide

```
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

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

Session 5

Operators and Expressions (Lab)

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);
}
```

Session 5

Operators and Expressions (Lab)

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 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

Session 5

Operators and Expressions (Lab)

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

Lab Guide

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
```

Session 5

Operators and Expressions (Lab)

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’ :

Session 5

Operators and Expressions (Lab)

```
#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)));
}
```

Lab Guide

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
```

Session 5

Operators and Expressions (Lab)

Part II: For the next 30 Minutes:

1. Solve the following expression:

$10 * 3 ^ 6 * 6 + 5 - 2 \text{ AND } (2 * 2 + 6 / 3 > 1 \text{ OR } 2 > 8)$

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. $x = (\text{int})3.8 + 3.3;$
- 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;
```

Session 5

Operators and Expressions (Lab)

Lab Guide



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;
```

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. Phiên này đề cập đến các thao tác nhập và xuất cơ bản.

Usually, input and output (I/O) form an important part of any program. Thông thường, nhập và xuất (I/O) là một phần quan trọng của bất kỳ chương trình nào. To do anything useful, your program needs to be able to accept input data and report back your results. Để làm được điều gì hữu ích, chương trình của bạn cần có khả năng nhận dữ liệu đầu vào và báo cáo lại kết quả. In C, the standard library provides routines for input and output. Trong ngôn ngữ C, thư viện tiêu chuẩn cung cấp các hàm cho nhập và xuất. The standard library has functions for I/O that handle input, output, and character and string manipulation. Thư viện tiêu chuẩn có các hàm I/O để xử lý nhập, xuất, và thao tác với ký tự và chuỗi. In this lesson, all the input functions described read from standard input and all the output functions described write to standard output. Trong bài học này, tất cả các hàm nhập được mô tả sẽ đọc từ đầu vào tiêu chuẩn và tất cả các hàm xuất sẽ ghi ra đầu ra tiêu chuẩn. Standard input is usually the keyboard. Đầu vào tiêu chuẩn thường là bàn phím. Standard output is usually the monitor (also called the console). Đầu ra tiêu chuẩn thường là màn hình (còn gọi là bảng điều khiển). Input and output can be rerouted from or to files instead of the standard devices. Nhập và xuất có thể được chuyển hướng từ hoặc đến các tệp thay vì các thiết bị tiêu chuẩn. The files may be on a disk or on any other storage medium. Các tệp có thể nằm trên đĩa hoặc trên bất kỳ phương tiện lưu trữ nào khác. The output may be sent to the printer also. Kết quả đầu ra cũng có thể được gửi đến máy in.

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

6.1 The Header File <stdio.h>

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

This is a preprocessor command. Đây là một lệnh tiền xử lý. In standard C, the # should be in the first column. Trong ngôn ngữ C tiêu chuẩn, dấu # phải nằm ở cột đầu tiên. stdio.h is a file and is called the header file. stdio.h là một tệp và được gọi là tệp tiêu đề. It contains the macros for many of the input/output functions used in C. Nó chứa các macro cho nhiều hàm nhập/xuất được sử dụng trong 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. Các hàm printf(), scanf(), putchar(), và getchar() được thiết kế sao cho chúng yêu cầu các macro trong stdio.h để thực thi đúng.

Session 6

Input and Output in ‘C’

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. Các hàm này được gọi là hàm định dạng. They can read and print data in various pre-specified formats which are under the user's control. Chúng có thể đọc và in dữ liệu theo các định dạng được chỉ định trước dưới sự kiểm soát của người dùng. Format specifiers specify the format in which the values of the variables are to be input and printed. Các định dạng chỉ định xác định định dạng mà giá trị của các biến sẽ được nhập vào và in ra.

6.2.1 `printf()`

You are already familiar with this function, since it has been used in the earlier sessions. Bạn đã quen thuộc với hàm này, vì nó đã được sử dụng trong các buổi học trước. We shall have a detailed look at the function here. Chúng ta sẽ xem xét chi tiết hàm này ở đây. The function `printf()` is used to display data on the standard output – console. Hàm `printf()` được sử dụng để hiển thị dữ liệu trên đầu ra chuẩn – bảng điều khiển. The general format of the function is: Định dạng chung của hàm là:

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

The argument list consists of constants, variables, expressions or functions separated by commas.

Danh sách đối số bao gồm các hằng số, biến, biểu thức hoặc hàm được ngăn cách bằng dấu phẩy.

There must be one format command in the control string for each argument in the list. Phải có một lệnh định dạng trong chuỗi điều khiển cho mỗi đối số trong danh sách. The format commands must match the argument list in number, type, and order. Các lệnh định dạng phải khớp với danh sách đối số về số lượng, loại và thứ tự. The control string must always be enclosed within double quotes (" "), which are its delimiters. Chuỗi điều khiển phải luôn được đặt trong dấu ngoặc kép (" "), là dấu phân cách của nó. The control string consists of one or more of three types of items which are explained below: Chuỗi điều khiển bao gồm một hoặc nhiều mục thuộc ba loại được giải thích dưới đây:

- **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. Các lệnh định dạng xác định cách mà các mục dữ liệu trong danh sách đối số sẽ được hiển thị. A format command begins with a % sign and is followed by a format code appropriate for the data item. Một lệnh định dạng bắt đầu với dấu % và được sau bởi mã định dạng phù hợp với mục dữ liệu. % is used by the `printf()` function to identify conversion specifications. Dấu % được sử dụng bởi hàm `printf()` để xác định các thông số chuyển đổi. The format commands and the data items are matched in order and typed from left to right. Các lệnh định dạng và mục dữ liệu được khớp theo thứ tự và nhập từ trái sang phải. One format code is required for every data item that has to be printed. Cần một mã định dạng cho mỗi mục dữ liệu phải được in ra.
- **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	<code>printf()</code>	<code>scanf()</code>
Single Character	%c	%c
String	%s	%s

Session 6

Input and Output in 'C'

Concepts

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 (số chữ số trong số nguyên).
%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 (1 chữ số bên trái dấu thập phân và 6 chữ số bên phải dấu thập phân).

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: Vì %, \ và " có những công dụng đặc biệt trong chuỗi điều khiển, nếu chúng cần được chèn vào như một phần của chuỗi văn bản và in ra màn hình, chúng phải được sử dụng như đã thấy trong Bảng 6.3. These symbols are interpreted differently by the printf() function unless escaped correctly. Các ký hiệu này sẽ được hàm printf() giải thích khác đi nếu không được thoát đúng cách. To print them as literal characters, special escape sequences are used. Để in chúng như các ký tự thực, cần sử dụng các chuỗi thoát đặc biệt. For example, to print a percentage sign, you need to use %% in the control string. Ví dụ, để in ký hiệu phần trăm, bạn cần sử dụng %% trong chuỗi điều khiển.

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

Table 6.3: Control String Special Characters

Session 6

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;
```

Session 6

Input and Output in 'C'

Concepts

```
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. Các lệnh định dạng có thể có các bộ sửa đổi để điều chỉnh các thông số chuyển đổi cơ bản. The following are valid modifiers acceptable in the printf() statement. Dưới đây là các bộ sửa đổi hợp lệ có thể chấp nhận trong câu lệnh printf(). If more than one modifier is used, then they must be in the same order as given below. Nếu sử dụng nhiều hơn một bộ sửa đổi, chúng phải được đặt theo thứ tự như dưới đây.

'-' 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). Bộ sửa đổi này có thể được sử dụng với kiểu float, double hoặc mảng char (chuỗi). The modifier is written as .m where m is an integer. Bộ sửa đổi được viết dưới dạng .m, trong đó m là một số nguyên. 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. Nếu sử dụng với kiểu dữ liệu float hoặc double, chuỗi số chỉ ra số lượng tối đa chữ số sẽ được in bên phải dấu thập phân. When used with a string, it indicates the maximum number of characters to be printed. Khi được sử dụng với chuỗi, nó chỉ ra số lượng ký tự tối đa sẽ được in.

Session 6

Input and Output in 'C'

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 off 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);
```

Session 6

Input and Output in 'C'

Concepts

```
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. Chúng tôi đã sử dụng [và] để chỉ ra nơi mà trường bắt đầu và nơi nó kết thúc. When we use %d with no modifiers, we see that it uses a field with the same width as the integer. Khi chúng ta sử dụng %d mà không có bộ sửa đổi, chúng ta thấy rằng nó sử dụng một trường có cùng chiều rộng với số nguyên. When using %10d, we see that it uses a field 10 spaces wide and the number is right-justified, by default. Khi sử dụng %10d, chúng ta thấy rằng nó sử dụng một trường rộng 10 khoảng trắng và số được căn bên phải theo mặc định. If we use the – modifier, the number is left-justified in the same field. Nếu chúng ta sử dụng bộ sửa đổi - , số sẽ được căn bên trái trong cùng một trường. If we use the 0 modifier, we see that the number is padded with 0 instead of blanks. Nếu chúng ta sử dụng bộ sửa đổi 0, chúng ta thấy rằng số được lấp đầy bằng 0 thay vì khoảng trắng.

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()
{
```

Session 6

```
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:
[555.55000]
In exponential form without any modifier:
[5.555500e+02]
In float form with - modifier:
[555.55000]
In float form with digit string 10.3 as modifier
[555.550]
```

Session 6

Input and Output in 'C'

Concepts

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. Trong phiên bản mặc định của %f, chúng ta thấy có sáu chữ số thập phân, và quy định mặc định của %e là một chữ số bên trái dấu thập phân và sáu chữ số bên phải dấu thập phân. 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. Hãy lưu ý rằng trong hai ví dụ cuối cùng, số chữ số bên phải dấu thập phân là 3, điều này khiến cho kết quả được làm tròn.

Now let us see how modifiers can be used with strings. Bây giờ, hãy cùng xem cách sử dụng các bộ điều chỉnh với chuỗi. Notice how the field is expanded to contain the entire string. Lưu ý rằng trường được mở rộng để chứa toàn bộ chuỗi. Also, note how the precision specification .4 limits the number of characters to be printed. Cũng hãy lưu ý cách quy định độ chính xác .4 giới hạn số ký tự được in ra.

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.");
}
```

Session 6

Input and Output in 'C'

```

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. Các ký tự mà chúng ta gõ trên bàn phím không được lưu trữ dưới dạng ký tự. Instead, they are stored as numbers in the ASCII (American Standard Code for Information Interchange) code format. Thay vào đó, chúng được lưu trữ dưới dạng số trong định dạng mã ASCII (Mã tiêu chuẩn Hoa Kỳ để trao đổi thông tin). The values that a variable holds are interpreted as a character or a numeric depending on the type of the variable that holds it. Các giá trị mà một biến nắm giữ được diễn giải là ký tự hoặc số tùy thuộc vào loại biến chứa nó. The following example demonstrates this: Ví dụ dưới đây minh họa điều này:

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);
}

```

Session 6

Input and Output in ‘C’

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. Kết quả này minh họa việc sử dụng các chỉ định định dạng và cách giải thích mã ASCII. 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. Mặc dù các biến a và b đã được khai báo là biến int và char, chúng đã được in ra dưới dạng ký tự và số bằng cách sử dụng các chỉ định định dạng khác nhau. This feature of C gives flexibility in the manipulation of data. Tính năng này của C mang lại sự linh hoạt trong việc xử lý dữ liệu.

How can you use the `printf()` statement to print a string which extends beyond a line of 80 characters? (Làm thế nào bạn có thể sử dụng câu lệnh `printf()` để in một chuỗi vượt quá một dòng 80 ký tự?) You must terminate each line by a \ symbol as shown in the example below (Bạn phải kết thúc mỗi dòng bằng ký hiệu \ như trong ví dụ dưới đây):

Example 6

A sample output is shown below:

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.

Session 6

Input and Output in ‘C’

```
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?

The `scanf()` function uses non-printing characters like blanks, tabs, and new lines to determine when an input field ends and when a new input field begins. Nó sử dụng các ký tự không in như khoảng trắng, tab và dòng mới để xác định khi nào một trường nhập liệu kết thúc và khi nào một trường nhập liệu mới bắt đầu. It matches the format commands with the fields in the argument list in the same order in which they are specified, skipping over any whitespace characters in between. Nó khớp các lệnh định dạng với các trường trong danh sách đối số theo cùng một thứ tự mà chúng được chỉ định, bỏ qua bất kỳ ký tự khoảng trắng nào ở giữa. Therefore, the input can be spread over more than one line, as long as there is at least one tab, space, or newline between each input field. Do đó, dữ liệu đầu vào có thể được trải dài trên nhiều dòng miễn là có ít nhất một tab, khoảng trắng hoặc dòng mới giữa mỗi trường nhập liệu. It effectively skips white spaces and line boundaries to obtain the data. Nó bỏ qua hiệu quả các khoảng trắng và ranh giới dòng để thu thập dữ liệu.

Example 7:

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

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

Session 6

Input and Output in 'C'

Concepts

```
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;
```

Session 6

Input and Output in ‘C’

```

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 (được thực thi), then 21 will be assigned (sẽ được gán) 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 (sẽ bị bỏ qua).

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 (Chuỗi sẽ có độ dài không xác định, nhưng sẽ giới hạn ở 79 kí tự) (actually, 80 characters including the null character that is added at the end-thực tế là 80 kí tự bao gồm kí tự null được thêm vào cuối).

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. Để chấp nhận bất kỳ ký tự nào cho đến ký tự xuống dòng, chúng ta sử dụng mã định dạng %[^n], điều này có nghĩa là chuỗi sẽ chấp nhận bất kỳ ký tự nào ngoại trừ \n, tức là ký tự xuống dòng. The caret (^) implies that all characters except those following the caret will be accepted as valid input characters. Ký tự mũ (^) chỉ ra rằng tất cả các ký tự ngoại trừ những ký tự đứng sau ký tự mũ sẽ được chấp nhận là ký tự đầu vào hợp lệ.

Session 6

Input and Output in 'C'

Example 10:

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

Concepts

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`. Khi hàm `scanf()` được thực thi, một chuỗi có độ dài không xác định (nhưng không quá 79 ký tự) sẽ được nhập từ thiết bị đầu vào tiêu chuẩn và được gán cho `line`. There will be no restriction on the characters that compose the string, except that they all fit on one line. Sẽ không có bất kỳ giới hạn nào về các ký tự tạo thành chuỗi, ngoại trừ việc tất cả chúng đều phải vừa trên một dòng. 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.

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

Session 6

Input and Output in 'C'

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. Bất kỳ ký tự nào khác trong `scanf()`, không phải là một phần của mã định dạng trong chuỗi điều khiển, phải được nhập vào chính xác như vậy; nếu không, sẽ gây ra lỗi. This feature is used to accept comma (,) delimited input. Tính năng này được sử dụng để chấp nhận dấu vào phân tách bằng dấu phẩy (,).

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. Lưu ý rằng các dấu phẩy trong chuỗi chuyển đổi khớp với các dấu phẩy trong luồng đầu vào và do đó sẽ đóng vai trò là dấu phân cách.

White space characters in the control string are normally ignored, except that it causes problems with the `%c` format code. Các ký tự khoảng trắng trong chuỗi điều khiển thường bị bỏ qua, ngoại trừ việc chúng gây ra vấn đề với mã định dạng `%c`. If we use the `%c` specifier, then a space is considered a valid character. Nếu chúng ta sử dụng định dạng `%c`, thì một khoảng trắng được coi là một ký tự hợp lệ.

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. Giá trị 14 sẽ được gán cho x, ký tự ch có giá trị là ký tự cách (decimal 99), và y được gán giá trị của ký tự c, có giá trị là decimal 99.

Consider the following code:

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

Session 6

Input and Output in 'C'

Concepts

```
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. Ở đây, chúng ta có thể thấy c2 chứa một khoảng trắng vì đầu vào chứa ký tự trắng. To skip over such white space characters and read the next non-white space character, the conversion group %1s should be used. Để bỏ qua các ký tự trắng như vậy và đọc ký tự không phải trắng tiếp theo, nhóm chuyển đổi %1s nên được sử dụng.

```
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. Ngôn ngữ C không tự định nghĩa các hoạt động nhập và xuất. All I/O operations are performed by the functions available in the C function library. Tất cả các hoạt động I/O được thực hiện bởi các hàm có sẵn trong thư viện hàm C. C function library contains one distinct system of routine that handles these operations. Thư viện hàm C chứa một hệ thống riêng biệt các quy trình để xử lý các hoạt động này. That is: Điều đó là:

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. Một bộ đệm là một khu vực lưu trữ tạm thời, có thể ở trong bộ nhớ hoặc trên thẻ điều khiển cho thiết bị. 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. Trong I/O được đệm, các ký tự được nhập từ bàn phím được thu thập cho đến khi người dùng nhấn phím return hoặc enter, khi đó các ký tự sẽ được cung cấp cho chương trình, dưới dạng một khối.

Buffered I/O can be further subdivided into:

- Console I/O
- Buffered File I/O

Session 6

Input and Output in 'C'

Console I/O refers to operations that occur at the keyboard and the screen of your computer. Nhập và xuất console để cập đến các hoạt động diễn ra tại bàn phím và màn hình của máy tính của bạn.

Buffered File I/O refers to operations that are performed to read and write data onto a file. Nhập và xuất tệp có bộ đệm để cập đến các hoạt động được thực hiện để đọc và ghi dữ liệu vào một tệp. We will discuss the console I/O. Chúng ta sẽ thảo luận về nhập và xuất console.

In C, console is considered as a stream device (console được xem là 1 thiết bị luồng). The Console I/O functions (hàm) direct their operations (hướng hoạt động của chúng) to the standard input and output of the system (hệ thống).

The simplest (đơn giản) Console I/O functions are:

- `getchar()` – which reads (đọc) one (and only one) character (kí tự) from the keyboard.(bàn phím)
- `putchar()` – which outputs (xuất ra) a single character (kí tự duy nhất) on the screen.(ra màn hình)

6.3.1 `getchar()`

The function `getchar()` is used to read input data, a character at a time from the keyboard. Hàm `getchar()` được sử dụng để đọc dữ liệu đầu vào, từng ký tự một từ bàn phím. In most implementations of C, `getchar()` buffers characters until the user types a carriage return. Trong hầu hết các triển khai của C, `getchar()` lưu trữ các ký tự cho đến khi người dùng nhập phím carriage return. Therefore, it waits until the return key is pressed. Vì vậy, nó sẽ đợi cho đến khi phím return được nhấn. The `getchar()` function has no argument, but the parentheses must still be present. Hàm `getchar()` không có tham số, nhưng dấu ngoặc đơn vẫn phải có. It simply fetches the next character and makes it available to the program. Nó đơn giản chỉ lấy ký tự tiếp theo và làm cho nó có sẵn cho chương trình. We say that the function returns a value, which is a character. Chúng ta nói rằng hàm trả về một giá trị, đó là một ký tự.

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. Trong chương trình trên, biến `letter` được khai báo có kiểu `char` để nhận dữ liệu đầu vào là ký tự.

Please enter any character:

will appear on the screen. Bạn sẽ thấy nó xuất hiện trên màn hình. You enter a character, say S, through the keyboard, and press carriage return. Bạn nhập một ký tự, chẳng hạn như S, qua bàn phím và nhấn phím return. The function getchar() fetches the character entered by you and assigns the same to the variable letter. Hàm getchar() lấy ký tự bạn đã nhập và gán nó cho biến letter. Later it is displayed on the screen with the message. Sau đó, nó được hiển thị trên màn hình cùng với thông điệp.

The character entered by you is S .

6.3.2 putchar()

putchar() is the character output function (hàm xuất kí tự) in C, which displays a character on the screen at the cursor position (hiển thị ký tự trên màn hình tại vị trí con trỏ). This function requires an argument (Hàm này yêu cầu một đối số.). 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()
{
```

Session 6

Input and Output in 'C'

```
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 (Console I/O để cập đến các hoạt động diễn ra tại bàn phím và màn hình máy tính của bạn. Nó bao gồm các hàm được định dạng và không được định dạng.).
- The formatted (định dạng) I/O functions are printf() and scanf().
- The unformatted (không định dạng) 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 " (Chuỗi điều khiển của printf() và scanf() phải luôn có bên trong và "). The string consists of a set of format commands (Chuỗi bao gồm một tập hợp các lệnh định dạng). Each format command consists of a %, an optional set of modifiers and a type specifier (Mỗi lệnh định dạng bao gồm một %, một tập hợp các trình sửa đổi tùy chọn và một trình chỉ định kiểu.).
- 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 (Sự khác biệt giữa getchar() và putchar() là putchar() nhận đối số trong khi getchar() thì không.).

Session 6

Input and Output in 'C'



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]

Session 6

Input and Output in 'C'



Try It Yourself

Concepts

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.



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.