Chapter 10

INPUT / OUTPUT

10.1 OVERVIEW

In previous chapters, we have studied the basics of C programs. This lesson covers basic input and output features of C language. Usually input and output form an important part of any program. To be more interactive, a program needs to be able to accept data and show results.

In C, the standard input/output library provides functions to perform input and output operations. By standard input and output, we mean the keyboard and monitor respectively. In C, these input/output operations are performed by two standard input/output functions, these are printf( ) and scanf( ). These functions can be accessed by including the standard input/output library (stdio.h) in the program. Let us have an overview of standard input/output functions.

10.1.1 printf Function

To see results of program execution, we must have a way to specify what variables values should be displayed. The standard library function *printf* (pronounced as print-eff) is used for formatted output. It takes as arguments a format string and an optional list of variables to output. The values of variables are displayed according to the specifications in the format string.

The printf ( ) function will take the form:

printf(*format string*, *var1*, *var2*, *var3*, ...);

printf(*format string);*

The *format string* is a character string – nothing more and the variables are optional. The easiest way to understand this is by example.

The **signature** of a function describes the number and type of its arguments, and the return type of the function.

Example 1 **Write a program to calculate and print the area of a square.**

```
#include <stdio.h>
void  main( )
{
   int  height, width, area;
   height = 5;
   width = 4;
   area = height * width;
   printf("Area of Square = %d", area);
}
```

Here's the **output** of the program:

```
Area of Square = 20
```

In the above program, the first line is the variable declaration statement. In second and third lines, values are assigned to the variables *height* and *width*. Fourth line of code describes the arithmetic expression for calculating the area of the square and the result is assigned to the variable *area*. Fifth and the last line of code is the printf( ) statement, which displays result on the screen. In case of *printf*, the first parameter is always a string (e.g.,"Area of Square"), which should be enclosed in double quotes. This string is called the *format string*. Format string may include any number of format specifiers such as %d. The list of variables separated by commas, whose values are to be displayed in the result, will follow the format string.

10.1.2 Format Specifier

Format specifiers specify the format in which the value of a variable should be displayed on the screen. Format specifiers are specified in the format string.

For instance, in the above program the printf( ) statement contains the symbol %d, which is format specifier for the variable *area*. For different types of variables, different format specifiers are used. Here is the list of format specifiers:

Symbol	Data Type
%d	int, short
%f	float
%lf	double
%e	float, double (Exponential Notation)
%g	Floating point (%f or %e, whichever is shorter)
%c	char
%s	Character string
%u	unsigned short, unsigned int
%x	Unsigned hexadecimal integers
%o	Unsigned octal integer
%i	Integers
%ld	long integer

Example 2 **Write a program that adds two floating point numbers and shows their sum on the screen.**

```
#include <stdio.h>

void main(void)
{
        float var1, var2, res;

        var1 = 24.27;
        var2 = 41.50;
        res = var1 + var2;
        printf("%f + %f = %f", var1, var2, res);
}
```

Here's the *output* of the program

```
24.27 + 41.5 = 65.77
```

10.1.3 Field-width Specifier

In a C program, the number of columns used to display a value on the screen is referred to as *field-width*. Field-width specifiers describe the number of columns that should be used to print a value.

Formatting Integers

We simply need to add a number between the % and d of the %d format specifier in the printf format string. This number specifies the field-width or the number of columns to be used for the display of the value. The statement

printf("Area = %4d", area);

indicates that four columns will be used to display the value of *area*. Suppose the value of the variable *area* is 25. Two extra spaces will be padded before 25 on the screen to complete the length of 4. The output of the above statement will be as follows:

Area = □□25

Here, □ represents a blank space. This space will not be displayed as a printed character in actual output. In this way the value of 25, which requires two spaces to be displayed, will occupy four spaces (columns) on the screen. The reason is that the format specifier for area (%4d) allows spaces for four digits to be printed. Because the value of area is 25, therefore its two digits are *right justified*, preceded by two blank spaces.

The following table shows how values are displayed using different format specifiers.

Value	**Format Specifier**	**Displayed Output**	**Value**	**Format Specifier**	**Displayed Output**
786	%4d	□786	-786	%4d	-786
786	%5d	□□786	-786	%5d	□-786
786	%6d	□□□786	-786	%6d	□□-786
786	%1d	786	-786	%2d	-786

The last row of this table shows that C expands the field width if it is too small for the integer value displayed.

Formatting Floating Point Numbers

For format specification of floating point numbers, we must indicate both the total *field width* needed and the number of *decimal places* desired. The total field width should be large enough to accommodate all digits before and after the decimal point e.g., to display 15.245 and 0.12 the total field width should be six and four respectively. It should be noted that for numbers smaller than zero, a zero is always printed before the decimal point. Therefore the total field width should include a space for the decimal point as well as for the minus sign if the number can be negative.

The general form for the format specifier for a floating point value will be *%m.n*f, where *m* represents the total field width, and n represents the desired number of decimal place. For instance, the statement

```
printf("Height = 6.2f", height);
```

indicates that the total field width for the value of the variable *height* is 6, and the accuracy is of two decimal places. The value of *height* will be rounded off to two decimal places and will be displayed *right justified* in 6 columns. While being rounded off, if the third digit of the value's fractional part is 5 or greater, the second digit is increased by one otherwise the third digit is discarded.

Remember: A format specifier always begins with the symbol %

The following table shows how values are displayed using different format specifiers.

Value	Format Specifier	Displayed Output	Value	Format Specifier	Displayed Output
-25.41	%6.2f	-25.41	.123	%6.2f	□□0.12
3.14159	%5.2f	□3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	□□3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	□3.14159
.6789	%4.2f	0.69	-0.007	%4.2f	-0.01
-.007	%8.3f	□□-0.007	-0.007	%8.5f	-0.00700
-.007	%.3f	-0.007	-3.14159	%.4f	-3.1416

10.1.3 Escape Sequences

Escape sequences are characters which are specified in the format string of the *printf* statement in combination with a backslash (\). These cause an escape from the normal interpretation of a string so that the next character is recognized as having a special meaning. For example, consider the following program

Remember: Escape sequence characters always begins with a backslash (\)

Example 3 **Write a program that will demonstrate the use of escape sequences.**

```
#include <stdio.h>

void main(void)
{
     printf("Name\t\tRoll_No\t\tMarks");
     printf("\n---------------------------------- ");
     printf("\nAmir\t\t  78\t\t425");
     printf("\nTahir\t\t  23\t\t385");
}
```

Here's the output of the program

```
Name                    Roll No                 Marks
-----------------------------------------------------------------
Amir                    78                      425
Tahir                   23                      385
```

In the above program, we use two escape sequences. These are **\t** and **\n**. The escape sequence **\n** causes the text to print from the start of the next line, whereas **\t** inserts a tab space between two words. In addition to newline and tab escape sequences; there are some others as well. Here is a list of them:

Escape Sequence	**Purpose**
\n	New Line
\t	Tab
\b	Backspace
\r	Carriage Return (Enter Key)
\f	Form feed
\'	Single Quote
\"	Double Quote
\\	Backslash
\xdd	ASCII code in hexadecimal notation (each d represents a digit)
\ddd	ASCII code in octal notation (each d represents a digit)

We have discussed the purpose of first two escape sequences. The escape sequence \b causes the cursor to move one space left, the form-feed (\f) moves to the next page on printer. It is important to note that one can not display a single or double quote on the screen without using the escape sequences \' and \". The reason is that the format string of the *printf* function is enclosed in a double quote. When a double quote is specified in the format string, it is treated as the closing double quote. That's why, single and double quotes are always written with backslash. For example the statement

```
printf("Escape Sequence is a \"Cool\" feature of C.");
```

Here's the *output*

```
Escape Sequence is a "Cool" feature of C.
```

10.2 SCANF FUNCTION

Most of the programs are interactive in nature. Till now, we did not learn a way to write interactive programs. C is featured with a range of functions to accept user input in variety of forms. The *scanf* (pronounced as scan-eff) function is versatile as it is equally good for numeric as well as string input.

It takes as arguments a format string and a list of variables to hold the input values. Here is the syntax of *scanf* function:

scanf(*format string*, *&var1*, *&var2*, *&var3*, ...);

Let us consider the following program to understand the working of scanf function:

Example 4 **Write a program to convert the distance in kilometers into meters.**

```
#include <stdio.h>

void main(void)
{
      double meter, kilometer;
      // prompt the user to enter kilometers
      printf("Enter distance in kilometers >");
      // take input
      scanf("%lf", &kilometer);
      meter = kilometer * 1000;
      printf("\n%lf kilometers = %lf meters",
      kilometer, meter);
}
```

Here's the sample *output* of the program

```
Enter distance in kilometers >40.5
40.500000 kilometers = 40500.000000 meters
```

In the above program, first line is the declaration of variables *meter* and *kilometer*. The next executable statement is *printf*, which displays a message for the user to enter distance in kilometers. The next is the *scanf* statement. When the program reaches this line of code, the flow of execution stops until the user enters a value. The format string of *scanf* is "%lf" which tells the scanf what kind of data to copy into variable *kilometer*. The format string of scanf consists of a list of format specifiers only; no other value or text can be specified in it.

Instead of the variable name, the *scanf* requires address of the variable to store the input value into it.

Notice that in the call to scanf, the name of variable *kilometer* is preceded by an ampersand character (&). In C, & is actually the *address of* operator. In scanf, the *address of* operator (&) tells the scanf funciton the address of the variable where the input value is to store. If & is omitted, the scanf will not be able to locate the variable in memory, hence it will be unable to store the value into the variable and the program will find a garbage value in the variable.

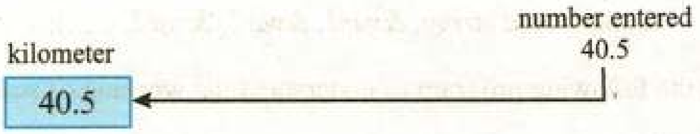

Fig. 10.1 Input value is stored in variable

10.3 CHARACTER INPUT

In C, there are many functions to accept character input. The versatile *scanf* can also be used for this purpose. But *scanf* requires pressing the return key at the end of input value. In some cases, it is desirable to input characters without pressing the return key. For example, in a game while controlling the movement of a space ship through arrow keys we can't afford to press return key each time after pressing an arrow key. To overcome such situations, C is equipped with many other functions specialized for character input. *getch* and *getche* are examples of such functions. These are part of the conio (console input output) library.

10.3.1 getch and getche Functions

The *getch* and *getche* functions are very handy in character manipulation. In contrast to the *getch* function which does not echo the character typed, the *getche* function echo the typed character. Both of these functions do not accept any argument. However they return the typed character to the calling function. One does not need to press the return key (ENTER key) after typing the character. The moment a character is typed, it is imidiately returned by the function to the calling module.

Note it that the character constants are always specified within single quotations e.g. 'a', '@', '7', etc.

Example 5 **Write a program that displays the ASCII code of the character typed by the user.**

```
#include <stdio.h>
#include <conio.h>
void main( )
{
     char ch;

     printf("Please type a character :");
     ch = getche( );
     printf("\nThe ASCII code for \'%c\' is %d", ch, ch);
}
```

Here's the **output** of the program

```
Please type a character :a
The ASCII code for 'a' is 97
```

Note it that, in this program we include a new header file *conio.h*. This file contains the definition of functions getch() and getche(). In this program, the statement

ch = getche();

can be replaced with the statement

ch = getch();

In the later case, the typed character will not be shown on the screen and the output will be as follows:

```
Please type a character :
The ASCII code for 'a' is 97
```

When the 3rd line of the program is executed, it waits for a character to be typed. As soon as a character is typed, the very next line executes imidiately without waiting for the return key to be typed. And the function getche() returns the typed character to the main function, where it is assigned to the variable **ch**.

Exercise 10c

1. Fill in the blanks:
   (i) The_______________ function does not display characters on the output screen.
   (ii) _____________ is an input function.
   (iii) %x is a format specifier for __________________.
   (iv) Escape sequences always begins with a _______________.
   (v) The printf function is defined in ____________________.
   (vi) The ASCII code for *Escape* key is ____________________.
   (vii) The escape sequence ________ represents the carriage return.
   (viii) There are total __________ columns on the output screen.
   (ix) The symbol for address of operator is _____________.
   (x) \dddd is used to print ASCII code in _____________ notation.

2. Choose the correct option:
   (i) The function getche() is defined in:
   a) stdio.h b) string.h
   c) math.h d) conio.h
   (ii) The escape sequence for backslash is:
   a) \ b) \b
   c) \\ d) \t
   (iii) The format specifier %u is used for:
   a) integer b) unsigned short
   c) unsigned float d) unsigned long int
   (iv)
   a) Arithmetic overflows b) Arithmetic underflow
   c) Truncation d) Round off
   (v) The symbol '=', represents:
   a) Comparison operator b) Assignment operator
   c) Equal-to operator d) None of these
   (vi) Which of the following operators has lowest precedence?
   a) ! b) +
   c) = d) ==
   (vii) Relational operators are used to:
   a) Establish a relationship among variables
   b) Compare two values
   c) Construct compound condition
   d) Perform arithmetic operations
   (viii) C is a strongly typed language, this means that:
   a) Every program must be compiled before execution
   b) Every variable must be declared before it is being used
   c) The variable declaration also defines the variable

d) Sufficient data types are available to manipulate each type of data

(ix) The logical not operator, denoted by !, is a:

a) Ternary operator b) Uniary operator
c) Binary operator d) Bitwise operator

(x) a += b is equivalent to:

a) b += a b) a =+ b
c) a = a + b d) b = b + a

3. Write T for true and F for false statement:

(i) printf and scanf are standard identifiers.
(ii) In C language you must declare all variables before using them.
(iii) Standard data types are not predefined in C language.
(iv) The double data type required 4 bytes in memory.

4. What do we mean by standard input and output? Illustrate the use of printf() and scanf() functions.

5. Illustrate the difference between format specifiers and field-width specifiers with examples.

6. Define the term 'escape sequence'. List names and uses of any five escape sequences.

7.

a) Show the output displayed by the following program when the data entered are 10 and 15.

```
#include <stdio.h>
void main()
{
        int m, n;
        printf("Enter two numbers(separated by
        comma):");
        scanf("%d %d", m, n);
        m = m + 10;
        n = 5 * m;
        printf("m = %d\t\t\t n = %d\n", m, n);
```

b) Show the contents of memory (for variables `'a'` and `'b'`) before and after the execution of the above program.

c) Write the program in example 5 using *scanf* function.

8.

a) Show how the value -17.246 would be printed using the formats %8.4f, %8.3f, %8.2f, %8.1f, %8.0f, and %0.2f.

b) Assuming x (type double) is 21.335 and y (type int) is 200, show the output of the following statements (on paper). For clarity, use the symbol □ to denote a blank space.

```
printf("x is %6.2f \t y is %4d\n", x, y);
printf("y is %d\n", y);
printf("x is %.1f\n", x);
```

c) If the variables a, b, and c are 307, 408.558 and -12.31, respectively, write a statement tha t will display the following line: (for clarity, the symbol □ shows a blank space)

□□307□□□□□408.56□□□□-12.3

9. Write a program that asks the user to enter the radius of a circle and then computes and displays the circle's area. Use the formula

area = PI × radius × radius

where PI is the constant value of 3.14159. (Note: Define a constant macro PI with #define directive)

10. Write a program that stores the values 'A', 'U', 3.456E10 and 50 in separate memory cells. Your program should get the first three values as input data, but use an assignment statement to store the last value.

11. Write a program that converts a temperature in degrees Fahrenheit to degrees Celsius. For conversion, use the following formula

celsius = 5/9 (fahrenheit – 32)

12. Write a program that takes a positive number with a fractional part and rounds it to two decimal places. For example, 25.4851 would round to 25.49, and 62.4431 would round to 32.44.