Chapter 3

Input-Output In C

There are three main functions of any program- it takes data as input, processes this data and gives the output. The input operation involves movement of data from an input device (generally keyboard) to computer memory, while in output operation the data moves from computer memory to the output device (generally screen). C language does not provide any facility for input-output operations. The input output is performed through a set of library functions that are supplied with every C compiler. These functions are formally not a part of the language but they are considered standard for all input-output operations in C. The set of library functions that performs input-output operations is known as standard I/O library.

There are several header files that provide necessary information in support of the various library functions. These header files are entered in the program using the #include directive at the beginning of the program. For example if a program uses any function from the standard I/O library, then it should include the header file stdio.h as-

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

Similarly there are other header files like math.h, string.h, alloc.h that should be included when certain library functions are used.

In this chapter we'll discuss about the input functions scanf( ) and getchar( ) and the output functions printf( ) and putchar( ). There are several other input-output functions that will be discussed in further chapters.

A simple method for taking the data as input is to give the value to the variables by assignment statement. For example-

```
int basic = 2000;
char ch = 'y';
```

But in this way we can give only particular data to the variables.

The second method is to use the input function scanf( ), which takes the input data from the keyboard. In this method we can give any value to the variables at run time. For output, we use the function printf( ).

3.1 Conversion Specifications

The functions scanf( ) and printf( ) make use of conversion specifications to specify the type and size of data. Each conversion specification must begin with a percent sign (%). Some conversion specification are as given below-

%c	-	a single character

%d	-	a decimal integer
%f	-	a floating point number
%e	-	a floating point number
%g	-	a floating point number
%lf	-	long range of tloating point number (for double data type)
%h	-	a short integer
%o	-	an octal integer
%x	-	a hexadecimal integer
%i	-	a decimal, octal or hexadecimal integer
%s	-	a string
%u	-	an unsigned decimal integer

The modifier h can be used before conversion specifications d, i, o, u, x to specify short integer and the modifier l can be used before them to specify a long integer. The modifier l can be used before conversion specifications f, e, g to specify double while modifier L can be used before them to specify a long double. For example %ld, %hd, %Lf, %hx are valid conversion specifications.

3.2 Reading Input Data

Input data can be entered into the memory from a standard input device (keyboard). C provides the scanf( ) library function for entering input data. This function can take all types of values (numeric, character, string) as input. The scanf( ) function can be written as-

scanf("control string" , address1, address2, ...);

This function should have at least two parameters. First parameter is a control string, which contains conversion specification characters. It should be within double quotes. The conversion specification characters may be one or more; it depends on the number of variables we want to input. The other parameters are addresses of variables. In the scanf( ) function at least one address should be present The address of a variable is found by preceding the variable name by an ampersand (&) sign. This sign is called the address operator and it gives the starting address of the variable name in memory A string variable is not preceded by & sign to get the address.

Some examples of scanf( ) function are as-

```
#include<stdio.h>
main( )
{
    int marks;
    ............
    scanf("%d",&marks);
    ............
}
```

In this example, the control string contains only one conversion specification character %d, which implie that one integer value should be entered as input. This entered value will be stored in the variable mark

```
#include<stdio.h>
main( )
{
    char ch;
    ............
    scanf("%c",&ch);
    ............
}
```

Here the control string contains conversion specification character %c, which means that a single character should be entered as input. This entered value will be stored in the variable ch.

```
#include<s
main( )
{
    float  height;
    ..............
    scanf("%f",&height);
    ..............
}
```

Here the control string contains the conversion speci which means that a floating point number should be entered as input. This entered va red in the variable height.

```
#include<stdio.h>
main( )
{
    char str[30] ;
    ..............
    scanf("%s",str);
    ..............
}
```

In this example control string has conversion specification character %s implying that a string should be taken as input. Note that the variable str is not preceded by ampersand(&) sign. The entered string will be stored in the variable str.

More than one value can also be entered by single scanf( ) function. For example-

```
#include<stdio.h>
main( )
{
    int  basic,da;
    ..............
    scanf("%d%d",&basic,&da);
    ..............
}
```

Here the control string has two conversion specification characters implying that two integer values should be entered. These values are stored in the variables basic and da. The data can be entered with space as the delimiter as-

    1500 1200

```
#include<stdio.h>
main( )
{
    int  basic;
    float  hra;
    char  grade;
    ..............
    scanf("%d  %f  %c",&basic,&hra,&grade);
    ..............
}
```

Here the control string has three conversion specifications characters %d, %f and %c, means that on integer value, one floating point value and one single character can be entered as input. These value are stored in the variables basic, hra and grade. The input data can be entered as-

1500 200.50 A

When more than one values are input by scanf( ), these values can be separated by whitespace characte like space, tab or newline (default). A specific character can also be placed between two conversio specification characters as a delimiter.

```
#include<stdio.h>
main( )
{
    int  basic;
    float  hra;
    ..........
    scanf("%d:%f",&basic,&hra);
    ..........
}
```

Here the delimiter is colon (:). The input data can be entered as-

1500:200.50

The value 1500 is stored in variable basic and 200.50 is stored in hra.

```
#include<stdio.h>
main( )
{
    int  basic;
    float  hra;
    ............
    scanf("%d,%f",&basic,&hra);
    ..........
}
```

Here the delimiter is comma (,). The input data can be entered as-

1500, 200.40

```
#include<stdio.h>
main( )
{
    int  day,month,year;
    int  basic;
    ..........
    scanf("%d-%d-%d",&day,&month,&year);
    scanf("$%d",&basic);
    ..........
}
```

Here if the data is entered as-

    24-5-1973
    $3000

Then 24 is stored in variable day, 5 is stored in variable month and 1973 is stored in variable year and 3000 is stored in variable basic.

If we include any spaces between the conversion specifications inside the control string, then they are just ignored.

```
#include<stdio.h>
main( )
{
    int  x,y,z;
    ..........
    scanf("%d  %d  %d",&x,&y,&z);
    ..........
}
```

If the data is entered as-

    12 34 56

Then 12 is stored in x, 34 is stored in y and 56 is stored in z.

3.3 Writing Output Data

Output data can be written from computer memory to the standard output device (monitor) using printf() library function. With this function all type of values (numeric, character or string) can be written as output. The printf( ) function can be written as-

    printf("control string", variable 1, variable 2,................);

In this function the control string contains conversion specification characters and text. It should be enclosed within double quotes. The name of variables should not be preceded by an ampersand(&) sign. If the control string does not contain any conversion specification, then the variable names are not specified. Some example of printf( ) function are as-

```
#include<stdio.h>
main( )
{
    printf("C  is  excellent\n");
}
```

Output:

```
    C is excellent
```

Here control string has only text and no conversion specification character, hence the output is only text.

```
#include<stdio.h>
main( )
{
    int  age;
    printf("Enter  your  age  :  ");
    scanf("%d",&age);
}
```

Here also printf does not contain any conversion specification character and is used to display a message that tells the user to enter his age.

```
#include<stdio.h>
main( )
{
    int  basic=2000;
    ...........
    printf("%d",basic);
    ...........
}
```

In this example control string contains a conversion specification character %d, which implies that ar integer value will be displayed. The variable basic has that integer value which will be displayed as output

```
#include<stdio.h>
main( )
{
    float  height=5.6;
    ..............
    printf("%f",height);
    ..............
}
```

Here control string has conversion specification character %f, which means that floating point numbe will be displayed. The variable height has that floating point value which will be displayed as output

```
#include<stdio.h>
main( )
{
    char  ch='$';
    ............
    printf("%c",ch);
    ...........
}
```

In the above example, the control string has conversion specification character %c, means that a singl character will be displayed and variable ch has that character value.

```
#include<stdio.h>
main( )
{
    char  str[30];
    ..............
    printf("%s",str);
    ................
}
```

Here control string has conversion specification character %s, implying that a string will be displaye and variable name str is a character array, holding the string which will be displayed.

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

```
{
    int basic=2000;
    printf("Basic Salary = %d",basic);
}
```

Output:

```
    Basic Salary = 2000
```

Here the control string contains text with conversion specification character %d. The text will be displayed as it is, and the value of variable basic will be displayed in place of %d.

```
#include<stdio.h>
main( )
{
    int b=1500;
    float h=200.50;
    char g='A';
    printf("Basic = %d , HRA = %f , Grade = %c",b,h,g);
}
```

Output:

```
    Basic = 1500 , HRA = 200.500000 , Grade = A
```

Here control string contains text with three conversion specification characters %d, %f and %c. %d is for integer value, %f is for floating point number and %c is for a single character.

```
#include<stdio.h>
main( )
{
    int num=10;
    printf("Octal equivalent of decimal %d = %o",num,num);
}
```

Output:

```
    Octal equivalent of decimal 10 = 12
```

Here the second conversion specification character is %o hence the octal equivalent of the decimal number stored in the variable num is displayed.

```
#include<stdio.h>
main( )
{
    int num=10;
    printf("Hex equivalent of decimal %d = %x",num,num);
}
```

Output:

```
    Hex equivalent of decimal 10 = A
```

Here the second conversion specification character is % x hence the hexadecimal equivalent of the decimal number stored in the variable num is displayed.

In chapter 2 we had studied about escape sequences. Here we'll see how we can use them in the printf statement. The most commonly used escape sequences used are '\n' and '\t'.

```
#include<stdio.h>
main( )
{
    int b=1500 ;
    float h=200.50 ;
    char g='A';
    printf("Basic = %d\nHRA = %f\nGrade = %c\n",b,h,g);
}
```

Output:

```
    Basic = 1500
    HRA = 200.500000
    Grade = A
```

'\n' moves the cursor to the beginning of next line. Here we have placed a '\n' at the end of control string also, it ensures that the output of the next program starts at a new line.

```
#include<stdio.h>
main( )
{
    int b=1500;
    float h=200.50;
    char g='A';
    printf("Basic = %d\tHRA = %f\tGrade = %c\n",b,h,g);
}
```

Output:

```
    Basic = 1500        HRA = 200.500000        Grade = A
```

'\t' moves the cursor to the next tab stop. Similarly we can use other escape sequences also. For example '\b' moves the cursor one position back, '\r' moves the cursor to the beginning of the current line and '\a' alerts the user by a beep sound. '\v' moves the cursor to the next vertical tab position(first column of the next line), and '\f' move the cursor to the next page. '\v' and '\f' are effective only when output is printed through a printer.

If we want to print characters like single quotes ('), double quotes (") or the backslash charcter (\ ), then we have to precede them by a backslash character in the format string.

For example-

printf("9 \\ 11 \\ 1978");	will print	9 \ 11 \ 1978
printf("She said, \"I have to go\". ");	will print	She said, "I have to go".

3.4 Formatted Input And Output

Formatted input and output means that data is entered and displayed in a particular format. Through format specifications, better presentation of result can be obtained. Formats for different specifications are as-

3.4.1 Format For Integer Input

%wd

Here 'd' is the conversion specification character for integer value and 'w' is an integer number specifying the maximum field width of input data. If the length of input is more than this maximum field width then the values are not stored correctly. For example-

```
scanf ("%2d%3d", &a, &b );
```

(i) When input data length is less than the given field width, then the input values are unaltered and stored in given variables.

Input-

6 39

Result-

6 is stored in a and 39 is stored in b.

(ii) When input data length is equal to the given field width, then the input values are unaltered and stored in given variables.

Input-

26 394

Result-

26 is stored in a and 394 is stored in b.

(iii) When input data length is more than the given field width, then the input values are altered and stored in the variable as -

Input-

269 3845

Result-

26 is stored in a and 9 is stored in b and the rest of input is ignored.

3.4.2 Format For Integer Output

%wd

Here w is the integer number specifying the minimum field width of the output data. If the length of the variable is less than the specified field width, then the variable is right justified with leading blanks. For example -

```
printf("a=%3d, b=%4d", a, b );
```

(i) When the length of variable is less than the width specifier.

Value of variables-

78 9

Output:

a	=		7	8	,	b	=				9

The width specifier of first data is 3 while there are only 2 digits in it, so there is one leading blank. The width specifier of second data is 4 while there is only 1 digit, so there are 3 leading blanks.

(ii) When the length of the variable is equal to the width specifier

Value of variables-

263 1941

Output:

```
a=263,b=1941
```

(iii) When length of variable is more than the width specifier, then also the output is printed correctly.

Value of variables-

2691 19412

Output:

```
a=2691,b=19412
```

```
#include<stdio.h>
main( )
{
    int  a=4000,b=200,c=15;
    printf("a  =  %d  \nb  =  %d  \nc  =  %d\n",a,b,c);
    printf("%a  =  %4d  \n%b  =  %4d  \nc  =  %4d\n",a,b,c);
}
```

The output of the first printf would be-

```
a = 4000
b = 200
c = 15
```

while the output of second printf would be-

```
a = 4000
b =  200
c =   15
```

3.4.3 Format For Floating Point Numeric Input

% wf

Here 'w' is the integer number specifying the total width of the input data (including the digits before and after decimal and the decimal itself). For example-

scanf("%3f %4f ", &x, &y);

(i) When input data length is less than the given width, values are unaltered and stored in the variables.

Input

5 5.9

Result

5.0 is stored in x and 5.90 is stored in y.

(ii) When input data length is equal to the given width, then the given values are unaltered and stored in the given variables.

Input

5.3 5.92

Result

5.3 is stored in x and 5.92 is stored in y

(iii) When input data length is more than the given width then the given values are altered and stored in the given variables as-

Input

5.93 65.87

Result

5.9 is store' in x and 3.00 is stored in y.

3.4.4 Format For Floating Point Numeric Output

%w.nf

Here w is the integer number specifying the total width of the input data and n is the number of digits to be printed after decimal point. By default 6 digits are printed after the decimal. For example-

printf("x = %4.1f, y = %7.2f ", x, y);

If the total length of the variable is less than the specified width 'w', then the value is right justified with leading blanks. If the number of digits after decimal is more than 'n' then the digits are rounded off.

Value of variables-

8 5.9

Output:

```
x= 8.0,y=   5.90
```

Value of variables-

25.3 1635.92

Output:

```
x=25.3,y=1635.92
```

Value of variables

15.231 65.875948

Output:

```
x=15.2,y=  65.88
```

3.4.5 Format For String Input

% ws

Here w specifies the total number of characters that will be stored in the string.

```
char str [8] ;
scanf ("%3s", str );
```

If the input is-

Srivastava

only first three characters of this input will be stored in the string, so the characters in the string will be-

'S', 'r', 'i', '\0'

The null character('\0') is automatically stored at the end.

3.4.6 Format For String Output

%w.ns

Here w is the specified field width. Decimal point and 'n' are optional. If present then 'n' specifies that only first n characters of the string will be displayed and (w - n) leading blanks are displayed before string.

(i) printf("%3s", "sureshkumar");

s	u	r	e	s	h	k	u	m	a	r

(ii) printf("%10s", "reeta");

					r	e	e	t	a

(iii) printf("%.3s", "sureshkumar ");

s	u	r

(iv) printf("% 8.3s", "sureshkumar ");

					s	u	r

(8 - 3 = 5 leading blanks)

3.5 Suppression Character in scanf( )

If we want to skip any input field then we specify * between the % sign and the conversion specification. The input field is read but its value is not assigned to any address. This character * is called the suppression character. For example-

scanf ("%d %*d %d", &a, &b, &c);

Input:

25 30 35

Here 25 is stored in 'a' , 30 is skipped and 35 is stored in the 'b'. Since no data is available for 'c' so it has garbage value.

scanf("%d %*c %d %*c %d", &d, &m, &y);

Input:

3/1/2003

Here 3 will be stored in d, then / will be skipped, 11 will be stored in m, again / will be skipped and finally 2003 will be stored in y.

```
#include<stdio.h>
main( )
{
    int a,b,c;
    printf("Enter three numbers :");
    scanf("%d %*d %d",&a,&b,&c);
    printf("%d %d %d",a,b,c);
}
```

Output:

```
Enter three numbers : 25 30 35
    25 35 25381
```

The variable c has garbage value.

3.6 Character I/O

3.6.1 getchar ( ) and putchar( )

These macros getchar( ) and putchar( ) can be used for character I/O. getchar( ) reads a single character from the standard input. putchar( ) outputs one character at a time to the standard output.

```
#include<stdio.h>
main( )
{
    char ch;
    printf("Enter a character : ");
    ch=getchar();
    printf("The entered character is : ");
    putchar(ch);
}
```

Output:

```
Enter a character : B
The entered character is : B
```

Exercise

Assume stdio.h is included in all programs.

```
(1) #define MSSG "Hello World\n"
    main( )
    {
        printf(MSSG);
    }

(2) main( )
    {
        printf("Indian\b is great\n");
        printf("New\rDelhi\n");
    }

(3) main( )
    {
        int a=11;
        printf("a = %d\t",a);
        printf("a = %o\t",a);
        printf("a = %x\n",a);
    }
```