# Unit - 2

# Chapter 5 Input and Output with C

## Chapter Outline

## 5.1 Introduction

A computer program probably would serve no useful purpose if a user cannot interact with the program. In most programming assignments it is necessary that the program reads input values from the user console and produces the intended useful output to the user. In this chapter, we will see how C manages various I/O operations.

Various input/output functions are available in C language. They are classified into two broad categories.

1. **Console Input/Output Functions:** Console Input Output functions help to read data from Keyboard and Print / Display data to the Monitor or screen.
2. **File Input/Output Functions:** File Input Output functions help to read data from Files on Disk and Write data to the Disk.

Let us discuss console input and output operations in this chapter.

## 5.2 Console Input and Output Operations

Reading the data from the input devices and displaying the output on the screen are the two main tasks of any program. Normally the standard input device (keyboard) is used to input data and the standard output device (screen/monitor) to display the results. When a program needs input, it gets the data through the input functions and sends the results obtained to the screen through the output functions. Therefore, the input / output functions are necessary to do this kind of job.

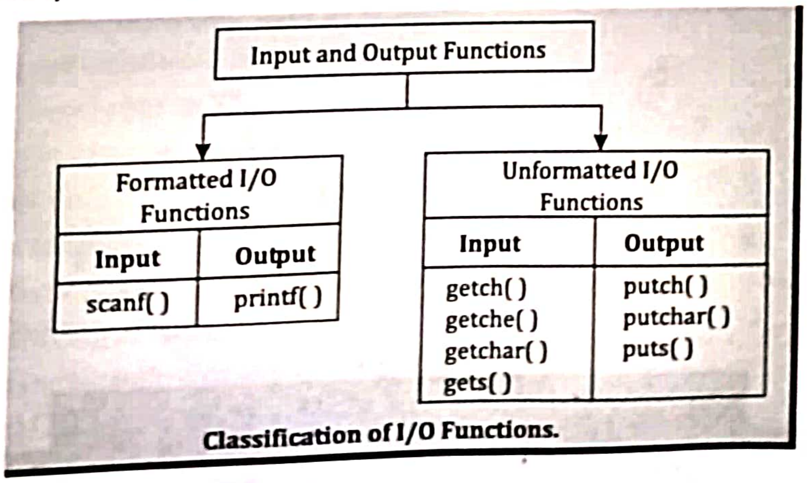
The input data entered by the user can be stored in the variables and can be used for processing. So far we have seen one method of providing input to the program. That is to assign values to variables through the assignment statements such as b = 10; f = 3.142; and so on.

C provides many functions for performing console I/O operations. These function permits the transfer of information between the computer's standard input and output devices (i.e., keyboard and monitor). Few of them give formatting control over input and output operations. Where as some of them doesn't allow to control the format of I/O operations.

To access these functions, it is necessary to include the standard I/O library header file. The header file stdio.h contains the declaration for these functions. Therefore, always include the header file stdio.h in C program before using these console I/O functions.

The Console Input/Output functions in C are classified into two types.

1. **Formatted I/O Functions**
2. **Unformatted I/O Functions**



**Classification of I/O Functions.**

### 1. Formatted Input/Output Functions

The formatted Input/Output functions that allow input and output operations to be performed in a specified and desired format. Formatting of I/O operations deals with some of the following issues.

1. How much field width is required to display the various values on the monitor?
2. How many decimal places are required to display the fractional part of a real number?
3. Should data values be left aligned or right aligned, and how much?
4. How much space between two data values is to be given?
5. How various types of data i.e. integer, character, and string can be used together in I/O operations?

The two most frequently used functions for formatted I/O are printf() and scanf(). The printf() is used to display the formatted data items on the standard output device normally the monitor whereas scanf() is used to read the formatted data input from the standard input device normally the keyboard. These functions are defined in the header file **stdio.h** and return EOF if there occurs an error or end of file.

The formatted I/O functions are used to read and write all types of data values. They require conversion symbol to identify the data type. These formatted functions allow the input to read from the keyboard in the format specified. Similarly, output can be displayed on to the screen according to the format specified.

**Example :** The formatted I/O functions are **scanf()** and **printf()**.

**Important Note**

**printf()** and **scanf()** functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in **"stdio.h"** which is a header file in C language.

We have to include **"stdio.h"** file in C program to make use of these printf() and scanf() library functions in C language.

### 2. Unformatted Input/Output Functions

Unformatted console I/O functions don't allow input and output to be formatted as per the user requirements. The unformatted I/O functions only work with the character data type. These functions do not require conversion symbol for identification of data types because they work only with the character data type. The unformatted I/O functions allow the user to input a single character or multiple characters using a standard input device such as keyboard and the output of a single character or multiple characters can be displayed on the standard output device such as screen.

**Unformatted I/O Functions are classified into two categories.**

1. **Unformatted Character I/O Functions:** The unformatted character I/O functions allow the user to input a single character using a standard input device such as keyboard and the output of a single character can be displayed on the standard output device such as screen.

   **Example: getchar(), putchar(), getch(), getche(), putch()**

2. **Unformatted String I/O Functions:** The unformatted string I/O functions allow the user to input a multiple characters or string using a standard input device such as keyboard and the output of a multiple characters or string can be displayed on the standard output device such as screen.

   **Example: gets() and puts()**

   A string is nothing, but a sequence of characters or group of chracters. gets() function is used to accept a string from the keyboard whereas puts() function is used to print a string on the monitor.

**What are the categories of Input and Output functions in C?**

The input/output functions are classified into two broad categories.

1. **Console Input/Output Functions :** Console Input Output functions help to read data from Keyboard and Print / Display data to the Monitor or screen.
2. **File Input/Output Functions :** File Input Output functions help to read data from Files on Disk and Write data to the Disk.

**What are the types of Console I/O functions ?**

The Console Input/Output functions in C are classified into two types.

1. **Formatted I/O Functions :** printf() and scanf() are examples
2. **Unformatted I/O Functions.:** getchar(), putchar(), getch(), getche(), putch(), gets() and puts() are examples.

## 5.3 Unformatted Input/Output Functions

### 5.3.1 Reading a Character using getchar()

The simplest input/output operation is reading a character from the 'standard input device' and writing a character to the 'standard output device'. Reading a single character from the standard input device can be done by using the function **getchar**. The **getchar( )** function reads one character at a time till the user presses the enter key. The **getchar( )** function does not require any argument. **getchar()** requires enter key to be pressed after the input of character.

**Syntax**

```
character_variable = getchar();
```

Where **character_variable** is any valid C variable name.

**Example**

```
char ch;
ch = getchar();
```

**Explanation**

When this statement is encountered in the program, the character entered from the keyboard will be displayed on the screen. After a single character is entered form the keyboard, the computer waits for the user to press the enter key. Once the enter key is pressed, the character entered is assigned to the character variable **'ch'**.

**Example** **The following code reads characters from keyboard one after another using getchar() until 'Enter key' is pressed.**

```
char ch;
while (ch ! = '\n')
{
        ch = getchar();
}
```

## 5.3.2 Writing a Character using putchar()

Writing a single character to the standard output device can be done by using the function putchar(). This function prints one character on the screen at a time. The putchar() function takes one argument and that argument is the character to the displayed.

**Syntax:**

```
putchar(variable-name);
```

Where, **variable-name** is a variable of type **char.**

**Example**

```
char ch = 'A'
putchar(ch);
```

**Explanation**

character varaible 'ch' is initialized with 'A';

will display the character 'A' on the screen.

**Example**

```
putchar('\n');
```

**Explanation**

will move the cursor to the beginning of next line.

**Program 1** The following program accepts a single character and displays the same by using getchar and putchar functions.

```
#include <stdio.h>
main()
{
        char ch;
        ch = getchar();
        putchar (ch);
}
```

**Output**

```
X
X
```

**Program 2** To demonstrate getchar and putchar

```
#include < stdio.h >
main()
{
        int i, j;
        char a, b;
        i = 65;             /* Assigns integer constant 65 to integer variable i */
        b = 'd';            /* Assigns character constant 'd' to character variable b */
        j = getchar();      /* Entered character gets stored in j */
        a = getchar();      /* Entered character gets stored in a */
        putchar(i);         /* displays 'A' as 'A's ASCII value is 65 */
        putchar(b);         /* displays alphabet 'd' */
        putchar(a);         /* displays entered character */
        putchar(j);         /* displays entered character */
}
```

**Output**

```
PQ
AdPQ
```

**Note on ASCII**

**ASCII** is the acronym for the **American Standard Code for Information Interchange**. It is a code for representing 128 English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for upper-case A is 65. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Technically, both the function i.e., getchar() and putchar() uses integer values to perform their respective operations, as character values are internally represented by their associated ASCII codes. For instance, the character value 'a' is equivalent to numeric value 97 as this number is the ASCII code for letter 'a'.

## 5.3.3 getch( ), getche( ) and putch( )

### 1. getch( ) and getche( )

The functions **getch( )** and **getche( )** are **unformatted** input functions. **getch( )** and **getche( )** are used to input only one character at a time through keyboard and they do not require **enter key** to be pressed after the input of character.

The difference between **getch( )** and **getche( )** is that **getch( )** does not display the input character while entering and **getche( )** displays (or) echoes the input character while entering the character. This added advantage of getch() could be proved beneficial in the application where user want to hide the input, for instance, password security.

**Example**

```
main()
{
        char ch;
        printf ("Enter a character:");
        ch = getch( );
        printf ("Entered character is %c", ch);
}
```

**Output**

```
Enter a character :
Entered character is : A
```

**Explanation**

Here, the character 'A' is entered by the user but getch( ) does not display the character typed.

If we use **getche( )** in place of **getch( ),** then we get the result as

**Example**

```
main()
{
        char ch;
        printf ("Enter a character:");
        ch = getche( );
        printf ("Entered character is %c", ch);
}
```

**Output**

```
Enter a character : A
Entered character is : A
```

**2. putch( )**

**putch( )** is used to display a character and the argument to this function is the character to be output.

**Example**

```
main( )
{
        char ch = 'A';
        putch(ch);
}
```

**Output**

```
A
```

**Explanation**

The above program outputs the character 'A' using **putch( ).** The functionality of this function is similar to that of **putchar( ).**

**What is the difference among getch(),getche(),getchar()?**

These all functions are used to get a single character from keyboard, but there are following differences.

**getch()** - It takes input, but does not echo the character on console, but requires enter key.

**getche()** - It takes input, echoes character and does not requires enter key.

**getchar()** - It takes input, echo character and requires enter key.

## 3.3.4 gets() and puts() functions

The limitation of unformatted character I/O functions is that they cannot handle more than one character at a time. Where as strings are used frequently in real life program. A string is nothing, but a sequence of characters. The functions which facilitates the transfer of string between the computer and the standard I/O devices are known as string I/O functions. Following functions can be used for handling strings I/O:

1. **gets()**: The gets() function receives a sequence of characters i.e., a string entered at the keyboard and store them in a variable (essentially as Array of type char) mentioned with it.
2. **puts()** : The puts() function, in contrast with gets(), writes a sequence of characters i.e., a string on the monitor .

The general syntax of gets() and puts() is:

**Syntax**

```
gets (character array variable);        puts(character array variable);
```

Where character Array is a valid C variable declared as an array of character type. The concept of arrays will be discussed in next chapters. The gets() function reads character from the standard input stream until a new line character ('\n') is read. The newline character is suppressed and a null ('\0') character is, then, appended in the end. This string is, then, stored in the memory address provided to by argument. That is why a string in C is also called as a character array terminated by a null character ('\0').

**Example**

```
char str[40];
gets(str);
```

It reads characters from the keyboard until a new line character is encountered and then appends a null character to the string automatically.

puts(string);

It is used to print the string on the monitor. The above statement will display a string on the monitor whatever is stored inside 'str' and will also causes the cursor to be moved in the next line.

**Program 3** **Write a program to read a string using gets( ) and display string using puts().**

```
# include <stdio.h>
main ( )
{
    char name[40], collegeName[40];
    printf("\n Enter your Name : ");
    gets (name);
    printf("\n Enter your College Name : ");
    gets(collegeName);
    printf ("\n Name is  : ");
    puts(name);
    printf ("\n College Name is  : ");
    puts(collegeName);
}
```

**Output**

```
Enter your Name : Indumathi
Enter your College Name : Jain College
Name is   : Indumathi
College Name is  : Jain College
```

## 5.4 Formatted Input / Output Functions

### 5.4.1 Formatted Output Statement (printf( ) function)

The printf() in one of the most important and useful functions to display data on monitor. We have seen the use of printf() for printing messages in the various example given previously in this book. For example, the statement printf(" C Programming"); will simple print this message on the monitor. Beside these text messages, a program frequently required numeric values and the value of other variables to be displayed on the screen.

The **printf( )** function is the **formatted** output function. This function prints all types of data values to the console. In many of our previous programs we have used the **printf** to print the string or values. In this section we will study the in-depth knowledge of printf function. The general form of printf statement is

**Syntax**

```
printf ("format string", list of variables);
```

where **format string** consists of

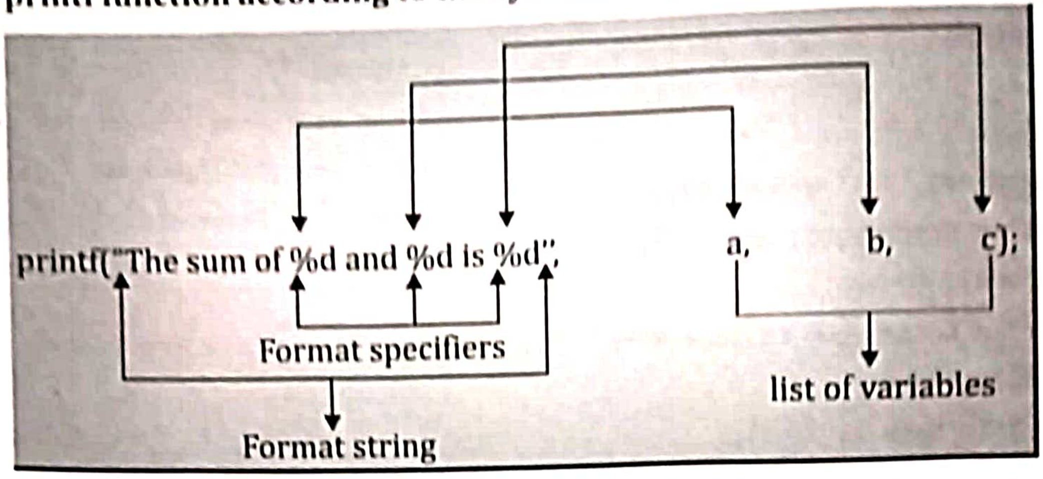
- Characters (or literal string) that are simply printed as they are
- Format Specifiers that begin with % sign. Example %d, %f, %s etc.
- Escape sequence characters that begin with backslash (\) character.

  **Example:** \n, \t, \b etc.

The **list of variables** are the variables whose values are formatted and printed according to the specifications of the format string. The list of variables should match in number, order and type with the format specifications.

**Return Value:** The function **printf** returns the number of characters that are displayed on the screen. But, most of the time we are not using the return value from the **printf()**.

**Example: The printf function according to the syntax is explained below.**



- The **format string** refers to a string in enclosed in double quotes that contain formatting information and $arg_1$, $arg_2$, ...., $arg_n$ are arguments (may be constants, variable, or other complex expressions) whose values are formatted and printed according to the specification of the format string.
- The format string contains the literal string, escape sequences and format specifiers like %d for integer, %f for float, %c for character etc.
- The format string usually consists of multiple characters. Except for double quotes, escape sequences and conversions specifiers, all characters with in a pair of double quotes will be treated as literal text (string or group of characters) and will be display as it is a on the monitor.

  The literal text is simply printed as entered in the format string. (The sum of, and, is are characters in the format string that are printed as they are)
- **Format specifiers** specify the type of data that can be displayed from the variables corresponding to the list of variables with the required format.
- In the above example, first format specifier %d corresponds to variable 'a' in the variables list, second %d to 'b' and third %d to 'c'.
- The printf() interpret the format string from left to right and start sending the characters to the standard output device. As soon as it encounters with the \ (backslash) (that is, the indication of the presence of as escape sequence), it takes action accordingly. Similarly when it encounters with % (conversion specifier) sign, it picks up the corresponding argument and prints its value according to the specified format. This process comes to an end, when it encounters the closing pair of double quotes.

**Example**

```
main( )
{
        printf ("Programming in C ");
        printf ("For Beginners");
}
```

**Output**

```
Programming in C For Beginners
```

**Explanation**

The above program prints the message enclosed within double quotes. The double quotes are delimiters and are not printed. Suppose if we want to print the second message in next line we can use escape sequence'\n'. The following example does the same.

**Example**

```
main( )
{
       printf ("Programming in C \n");
       printf ("For Beginners");
}
```

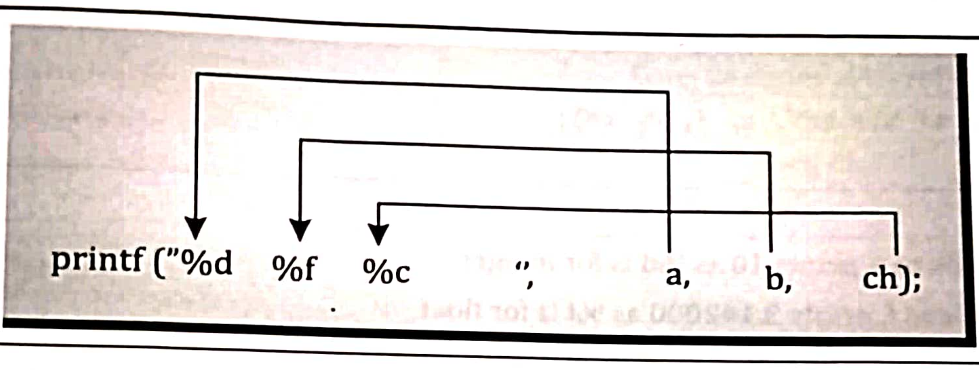
**Output**

```
Programming in C
For Beginners
```

**Explanation**

In the above example, we have used '**\n**' in first printf statement. The escape sequence '\n' is called **newline character.** This character takes the cursor to the beginning of the next line.

**Example**



**Explanation**

Here, first format specifier %d corresponds to a variable **a.** Second format specifier %f corresponds to a variable **b.** Third format specifier %c corresponds to ch. Therefore the variable **a, b** and **c** should be of type **int, float** and **char** respectively.

**Caution**

Argument list must be corresponding to the format specifiers mentioned in the format string i.e., number and the type of arguments must correspond to the conversion specifiers, otherwise unexpected result may come.

**Program 4** **The following program prints the sum of two numbers.**

```
main( )
{
      int a, b, c;
      a = 20;
      b = 30;
      c = a + b;
      printf ("The sum of %d and %d = %d", a, b, c);
}
```

**Output**

```
The sum of 20 and 30 = 50
```

**Explanation**

In the above program, the first statement declares 3 variables a, b, and c of type **int**. In the second and third statements, a is assigned to 20 and b is assigned to 30. The fourth statement assigns the sum of 20 and 30 (which is 50) to a variable c. The function printf prints the output as

The sum of 20 and 30 = 50

Here, first format specifier %d corresponds to a and therefore the value of a is printed in place of first %d, the second %d corresponds to b and therefore the value of b is printed in place of second %d, the last %d corresponds to c and therefore the value of c is printed in place of last %d.

**Program 5** **The following program demonstrates the printing of different data values.**

```
main( )
{
        int a = 10;
        float f = 3.142;
        double d = 8.525;
        char ch = 'A';
        printf ("%d %f %lf %c", a, f, d, ch);
}
```

**Output**

```
10  3.142000  8.525000  A
```

**Explanation**

Here, %d corresponds to a, prints 10 as %d is for integer
%f corresponds to f, prints 3.142000 as %f is for float
%lf corresponds to d, prints 8.525000 as %lf is for double
%c corresponds to ch, prints A as %c is for character

## 5.4.2 Format Specifiers

**Format specifiers** specify the type of data that can be displayed from the variables corresponding to the list of variables with the required format. The format specifiers should match the variables in number, order and type. If there are not enough variables or if they are of the wrong type, then the output result will be incorrect. The %d, %f and %c used in the printf( ) are called **Format Specifiers.** These are also called **conversion symbols or conversion specifications.** The following table lists the different types of format specifiers used in printf( ).

**What is Format Specifier? What is its use?**

A Format Specifier specifies type of data to be Read from Keyboard and type of data to be printed on console/monitor/screen.

C Format Specifiers are used in Input/Output functions to display data in a specified format so that it can be read easily and printed in desired format.

**Example:** %d for integer, %f for float, %s for strings, %c for character

| Format Specifier | Variable/Constant in the arguments list | Output |
|---|---|---|
| %d or %i | integer | Short signed integer |
| %c | character | Signed character or Unsigned character |
| %f | float | Signed float |
| %o | integer | Unsigned octal integer |
| %u | integer | Short unsigned integer |
| %x | integer | Unsigned hexadecimal integer [using lower case letters a-f] |
| %g | float | Floating point number in exponent form |
| %e | float | Floating point number (or) either in exponent form float form depending on value. |
| %X | integer | Unsigned hexadecimal integer. [Using upper case letters A-F]. |
| %ld or %li | integer | Long decimal integer |
| %lu | integer | Long unsigned decimal integer |
| %lf, %le, %lg | double | double |
| %Lf, %Le, %Lg | double | long double |
| %s | string | Sequence of characters |

## 5.4.3 Escape Sequence

Certain non printable characters, which are used in the printf() function are called as **Escape Sequences.** An escape sequence character begins with a backslash and is followed by one character. A backslash (\) along with some character give special meaning and purpose for that character.

The backslash (\) is called an escape character. It indicates that printf is supposed to do something out of the ordinary. When encountering a backslash in a string, the compiler looks ahead at the next character and combines it with the backslash to form an escape sequence. The escape sequence \n means newline. When a newline appears in the string output by a printf, the newline causes the cursor to position to the beginning of the next line on the screen. Some common escape sequences are listed below.

| Escape Sequence | Meaning |
|---|---|
| \n | New line |
| \b | Back space |
| \f | Form feed |
| \' | Single quote |
| \t | Horizontal Tab |
| \a | bell / alert |
| \r | Carriage return |
| \v | Vertical tab |
| \" | Double quote |
| \0 | NULL |

| Example | Output |
|---|---|
| printf ("Hello \n Hai"); | prints Hello<br>Hai |
| printf ("Hello \"Hai\" "); | prints Hello "Hai" |
| printf ("Hello \t Hai"); | prints Hello Hai |
| printf ("Hello \' Hai\' "); | prints Hell 'Hai' |
| printf ("Hello\b Hai"); | prints Hell Hai<br>[One character before\b is erased] |

The escape sequence '\r' takes the cursor to the beginning of the line in which it is currently plac '\a' alerts the user by generating the sound inside the computer. '\b' moves the cursor one positi to the left of its current position. '\f' advances the computer stationery attached to the printer to t top of the next page.

**Explain Escape Sequences with an example.**

Certain non printable characters, which are used in the printf() function are called as **Escape Sequences**. An escape sequence begins with a backslash '\' followed by a character or characters. The C compiler interprets any character followed by a '\' as an escape sequence. Escape sequences are used to format the output text and are not generally displayed on the screen. Each escape sequence has its own predefined function.

**Example:** \n is used to create a new line and place the cursor in the beginning of new line. Words that come after '\n' will be pushed to a new line..

## 5.4.4 Field Width Specification for Printing Integers

Field width can be specified in the format specification. The printf function uses this field width to know how many columns on the screen should be used while printing a value.

**Syntax**

| %WC | %-WC |
|---|---|
| right justified | left justified |

Where

- W indicates total number of columns required to print the value. It is also called the total width of the output.
- C is the format specifiers for the integer. C can be one of the format specifiers for integers like %d, %u, %lu, %x, %o etc.

Displaying the value from right to left is said to be **right justified** and displaying the value from left to right is said to be **left justified.**

**Example**

Suppose, a = 254 and field width specification is %5d (right justification). To print the value 254, totvvvally 5 columns are reserved out of which only 3 columns are used to print the value 254. The sample output looks like

$\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{2}\boxed{5}\boxed{4}$ **Right Justified**

If field width specification is %-5d (left justification), then the output looks like

$\boxed{2}\boxed{5}\boxed{4}\boxed{\phantom{0}}\boxed{\phantom{0}}$ **Left Justified**

Minus sign will be used for left justification

**Example**

If a = 345 and %2d is the field width specification. To print the value 345 it requires at least 3 columns, but we are reserved only 2 columns. In that case extra column will be automatically used to print the remaining digits of a number.

$\boxed{3}\boxed{4}\boxed{5}$

The output will be right justified. The column for digit 5 is automatically created.

**Note**

- If number of columns (W) are not sufficient to print a value, then additional columns are automatically created.
- If less columns are occupied with data, blanks will be shown in place holders. Blanks will appear on right if the text is Left Justified. Blanks will appear on Left if the text is Right Justified.

**Example**

| Statement | Output |
|---|---|
| printf ("%d", 1234); | $\boxed{1}\boxed{2}\boxed{3}\boxed{4}$ |
| printf ("%5d", 1234); | $\boxed{\phantom{0}}\boxed{1}\boxed{2}\boxed{3}\boxed{4}$ |
| printf ("%8d", 1234); | $\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{1}\boxed{2}\boxed{3}\boxed{4}$ |
| printf ("%2d", 1234); | $\boxed{1}\boxed{2}\boxed{3}\boxed{4}$ Additional columns are created automatically. |
| printf ("%-5d", 1234); | $\boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{\phantom{0}}$ |
| printf("%06d", 1234); | $\boxed{0}\boxed{0}\boxed{1}\boxed{2}\boxed{3}\boxed{4}$ The leading blanks will be filled with Zeros by placing the 0 before the field width specifier. |
| printf("%d",9876543210) | This is very long number and hence output will be junk value. Use %ld for long numbers instead of %d |
| printf("%ld",9876543210) | $\boxed{9}\boxed{8}\boxed{7}\boxed{6}\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{1}\boxed{0}$ |

## 5.4.5 Field Width Specification for Printing Decimal Numbers (Float)

The syntax of specifying field width for **float** values is shown below.

**Syntax:**

| %W.Xf | %-W.Xf |
|---|---|
| **right justified** | **left justified** |

Where

- **W** indicates total number of columns required to print the value. Which is nothing but tot field width.
- **X** is the number of columns used after the decimal part.
- **f** is the format specifier for float data type.

If we want to print the value in exponent form, then we should use below syntax

| %W.Xe | %-W.Xe |
|---|---|
| **right justified** | **left justified** |

If number of columns (W) are not sufficient to print a value, then additional columns are create automatically to print the number. By default, the output will be right justified. For left justificatio we should use – sign before W.

| Example | Output and Explanation |
|---|---|
| If a = 86.123<br>Field width specification is %7.2f,<br>printf("%7.2f",a); | `\| \| \|8\|6\|•\|1\|2\|`<br>We have reserved totally 7 columns to print 86.123 and we specified only 2 columns after decimal part. Therefore 3 is not printed. |
| If a = 86.123<br>Field width specification is %–7.2f,<br>printf("%-7.2f",a) | `\|8\|6\|•\|1\|2\| \| \|` |
| If a = 86.123<br>Field width specification is %–7.3f,<br>printf("%-7.3f",a) | `\|8\|6\|•\|1\|2\|3\| \|` |
| If a = 86.123<br>Field width specification is %12.2e,<br>printf("%12.2e",a) | `\| \| \| \| \|8\|•\|6\|1\|e\|+\|0\|0\|` |

**Example**

If a = 5432.123

| Statement | Output |
|---|---|
| printf ("%f", a); | $\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{\bullet}\boxed{1}\boxed{2}\boxed{3}\boxed{0}\boxed{0}\boxed{0}$ |
| printf ("%12.2f", a); | $\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{\bullet}\boxed{1}\boxed{2}$ |
| printf ("%-12.2f", a); | $\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{\bullet}\boxed{1}\boxed{2}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}$ |
| printf ("%3.1f", a); | $\boxed{5}\boxed{4}\boxed{3}\boxed{2}\boxed{\bullet}\boxed{1}$ Here, w = 3 and it is not sufficient. Extra 3 columns are automatically created and it is right justified. |
| printf("%e", a); | $\boxed{5}\boxed{\bullet}\boxed{4}\boxed{3}\boxed{2}\boxed{1}\boxed{2}\boxed{3}\boxed{e}\boxed{+}\boxed{0}\boxed{3}$ |
| printf ("%13.2e", a); | $\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{5}\boxed{\bullet}\boxed{4}\boxed{3}\boxed{e}\boxed{+}\boxed{0}\boxed{3}$ |
| printf ("%-13.2e", a); | $\boxed{5}\boxed{\bullet}\boxed{4}\boxed{3}\boxed{e}\boxed{+}\boxed{0}\boxed{3}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}$ |

## 5.4.6 Field Width Specification for Printing Strings and Characters

The syntax of specifying field width for strings and single character is shown below.

**Syntax**

| %W.Xs | %-W.Xs |
|---|---|
| **right justified** | **left justified** |

Where

- W indicates total number of columns reserved to print the string.
- X is the number of columns used to print the string in right justified way. The rest of characters are ignored.
- s is the format specifier for string (sequence of characters).

**Example**

| | Output |
|---|---|
| If a = "SKYWARD BOOKS"<br>Field width is %s<br>printf("%s",a); | $\boxed{S}\boxed{K}\boxed{Y}\boxed{W}\boxed{A}\boxed{R}\boxed{D}\boxed{\phantom{0}}\boxed{B}\boxed{O}\boxed{O}\boxed{K}\boxed{S}$ |
| If a = 'A' and %4c is the field width<br>printf("%4c",a); | $\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{A}$ |

| | |
|---|---|
| If a = 'A' and %-4c is the field width<br>printf("%-4c",a); | A ▢ ▢ ▢ |

We can truncate right side of a String using Field Width Specifiers as part of Format Specifiers. We ca
printf only first N character by using %(W.N)s notation with strings.

**Example**

If a = "SRIKANTH"

| Statement | Output |
|---|---|
| printf ("%s", a); | S R I K A N T H<br>The above statement prints the string (nothing special happens.) |
| printf ("%5s", a); | S R I K A N T H<br>The complete string will be printed if the string is bigger than the width mentioned. |
| printf ("%12s", a); | ▢ ▢ ▢ ▢ S R I K A N T H<br>statement prints the string, but print 12 characters. If the string is smaller the "empty" positions will be filled with "whitespace." (right justified) |
| printf ("%-12s", a); | S R I K A N T H ▢ ▢ ▢ ▢<br>statement prints the string, but prints at least 12 characters. The string in this case is shorter than the defined 12 character, thus "whitespace" is added at the end (left justified.) |
| printf ("%12.6s", a); | ▢ ▢ ▢ ▢ ▢ ▢ S R I K A N<br>12 columns are reserved but only first 6 characters will be displayed in right justified. The remaining 6 characters will be filled with white spaces. |
| printf ("%-12.6s", a); | S R I K A N ▢ ▢ ▢ ▢ ▢ ▢<br>12 columns are reserved but only first 6 characters will be displayed in left justified. The remaining 6 charaters will be filled with white spaces. |
| printf ("%12.0s", a); | ▢ ▢ ▢ ▢ ▢ ▢ ▢ ▢ ▢ ▢ ▢ ▢<br>12 columns are reserved but zero characters will be displayed. All the columns will be filled with white spaces. |

| | |
|---|---|
| printf ("%.6s", a);<br>printf ("%0.6s", a);<br>printf ("%-.6s", a); | \| S \| R \| I \| K \| A \| N \|<br>Only first 6 characters will be displayed. |

**Program 6** **To Demonstrate the Field Width Specifications**

```
#include <stdio.h>
main( )
{
        float f = 25.123;
        int a = 1234;
        char ch = 'A';
        printf ("\n Result 1: %f", f);
        printf ("\n Result 2: %10f", f);
        printf ("\n Result 3: %-10f", f);
        printf ("\n Result 4: %e", f);
        printf ("\n Result 5: %10.2f", f);
        printf ("\n Result 6: %-10.2f", f);
        printf ("\n Result 7: %d", a);
        printf ("\n Result 8: %12d", a); .
        printf ("\n Result 9: %-12d", a);
        printf ("\n Result 10: %2d", a);
        printf ("\n Result 11: %c", ch);
        printf ("\n Result 12: %4c", ch);
        printf ("\n Result 13: %-4c", ch);
}
```

**Output**

```
Result 1 :  25. 122999
Result 2 :     25. 122999
Result 3 :  25. 122999
Result 4 :  2.512300e+01
Result 5 :      25. 12
Result 6 :  25. 12
Result 7 :  1234
Result 8 :              1234
Result 9 :    1234
Result 10: 1234
Result 11 : A
Result 12 :   A
Result 13 : A
```

## 5.4.7 Formatted Input Statements (scanf( ) function)

As mentioned earlier, in order to write interactive program we must include some statements in a program that could be able to receive the data from user. In this context, we presented a couple of functions like getchar(), gets(), and getch() etc. But a programmer needs more flexibility in terms of:

1. Read the data from keyboard according to a specified format
2. Instruct the compiler to receive the particular type of value from the keyboard. For instance, integer value or floating point value.
3. Instruct the compiler to read the specified number of digits of a given number.
4. Reading mixed data types from the keyboard using single function.

But the use of unformatted input/output functions is restrict with the character values only. There is a need of more flexible and general function that could address the problems mentioned above.

The **scanf( )** function is the **formatted** input function. The **scanf( )** is used to accept the values of any data type. This function scans (or) accepts the values from the standard input device (keyboard) to the address of variables mentioned in the argument list.

**Syntax**

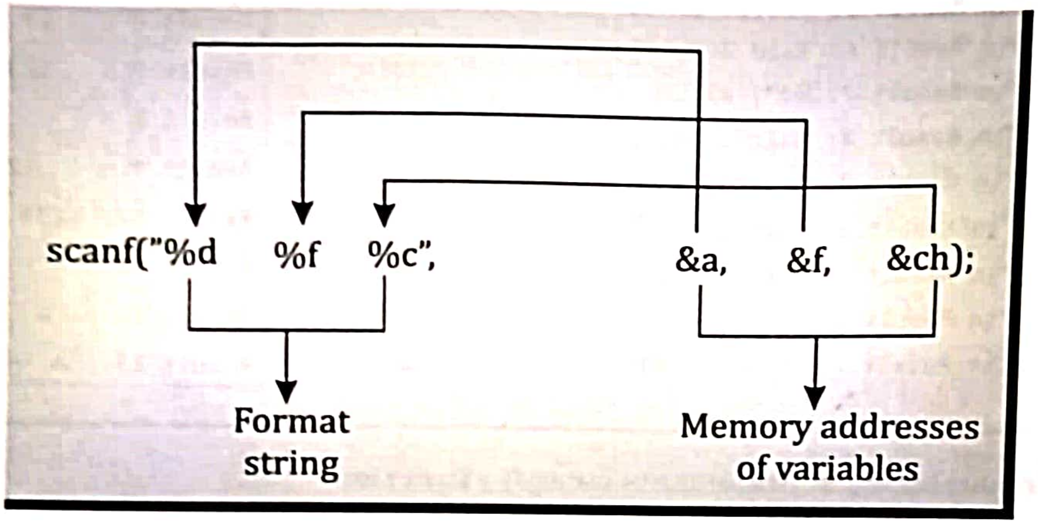
```
scanf("format string", &var₁, &var₂ .. &varₙ);
```

Where

- where **format string** consists of format specifier which is used to specify the data type of the valu being read.
- format specifier begin with % sign. Example %d, %f, %s etc.
- escape sequence characters that begin with \ character. Example \n, \t, \b etc.
- $var_1$, $var_2$ ... $var_n$ are the arguments passed to the function scanf. They are list of memory addresse of variables.

**Returns :** The scanf( ) function returns an integer value. This integer value is the number of elements tha have been entered successfully using the standard input device

**Example:** The format string is enclosed in double quotes and the second argument is the **list o memory address** of variables whose values are to be input. There is comma (,) separating the forma string and memory addresses list.



**Format String: :**

The format string consists of group of characters. The group of characters are different for each type of input data. In simple, format string consists of **format specifiers.** The format specifiers for **float, int** and **char** type variables whose values to be input are **%f, %d** and **%c** respectively.

The number of format specifiers enclosed within double quotes equal to the number of variables whose values are to be input.

**List of Memory Addresses :**

The second argument in **scanf( )** is the list of memory addresses of variables whose values are to be input. The values we input for different variables are copied to memory locations represented by those variables by accessing their memory addresses.

Memory address of a variable may be obtained by using an **ampersand(&)** before the variable name. **For example &a, &f** and **&ch** are memory addresses of variables **a, f** and **ch** respectively. The ampersand(&) is called **address operator.** In the **scanf( )** statement the role of '**&**' operator is to indicate the memory location of the variable, so that value read would be placed at that location.

In a very layman language, the above statement can be interpreted as an instruction to the complier to receive an integer value from the keyboard and store it in a variable named a. Where a must be an integer variable declared earlier. Once the value has been stored in the variable, it can be used anywhere for any purpose after this statement.

| Example | Statement |
|---|---|
| To input value for 1 integer variable<br>**int a;** | **scanf("%d",&a)**<br>Will read integer value from the keyboard, the entered value will be assigned to a |
| To input values for 3 integer variables<br>**int a, b,c;** | **scanf("%d %d %d",&a,&b,&c);**<br>Will read three integer values from the keyboard, first value will be assigned to a, second to b and third to c. |
| To input value for a character variable<br>**char ch;** | **scanf("%c",&ch);**<br>Will read character value from the keyboard, the entered value will be assigned to ch. |
| To input value for a float variable<br>**float f;** | **scanf("%f",&f);**<br>Will read float value from the keyboard, the entered value will be assigned to f |
| To input value for a double variable<br>**double d;** | **scanf("%lf",&d);**<br>Will read double value from the keyboard, the entered value will be assigned to d |
| To input value for one integer, one float, one double variable<br>**int a, float f; double d;** | **scanf("%d %f %lf",&a,&f,&d);**<br>Will read integer, float and double values from the keyboard, first value will be assigned to a, second to f and third to d. |

**Important note on scanf()**

scanf(), the complement of the printf(), can actually be used to read the different type of data from the keyboard in a specified format. Due to what, it is referred to as formatted input functions. Like printf(), scanf() also uses a format string to describe the format of the input, but with some little variations as given below:

1. It doesn't allow escape sequences in the format string.
2. It requires a special operator & called as "address of" to be prefix with the variable names
3. When multiple variables are entered in a single scanf(), they can be separated using white space character (i.e., blank space, tabs or new line character). White spaces in the format specifier itself are ignored it may be a space, or a tab, or even a newline character.

   **Example :** scanf("%d %d %f %ch \n", &a, &b, &f, &ch);

**Program 7** The following program accepts an integer and a float value and prints the same.

```
#include <stdio.h>
main( )
{
    int a;
    float f;
    printf("\n Enter an integer and float value:");
    scanf("%d %f", &a, &f);
    printf("\n The integer value entered is : %d", a);
    printf("\n The float value entered is : %f", f);
}
```

**Output**

```
Enter an integer and float value :
155   3.142
The integer value entered is : 155
The float value entered is : 3.142000
```

**Explanation**

In the above program we have used scanf( ) to read the values for a and f respectively. The statement is explained as follows.

$$\texttt{scanf(}\underbrace{\texttt{"\%d \%f"}}_{\text{Format string}}\texttt{,}\quad \underbrace{\texttt{\&a, \&f}}_{\text{variables with address}}\texttt{);}$$

- Each variable has an address in which the value will be stored. **&a** in a scanf function informs to store the value at the address of variable a. Similarly &f informs to store the value at the address of variable **f**.
- %d in format string corresponds to the variable a (a is of int type) and %f corresponds to the variable f (f is of float type).
- First it reads keyboard input as a integer number and assigns that value to the variable a. Then it reads float value and assigns that value to the variable

**Program 8** The following program illustrates the return value of scanf()

```
main( )
{
    int a, b, c, v;
    printf ("Enter values of a, b, & c \n");
    v = scanf ("%d %d %d", &a, &b, &c);
    printf ("\n Number of values successfully read is : %d", v);
}
```

**Output**

```
Enter values of a, b, & c
10 20 30
Number of values successfully read is : 3
```

**Explanation**

In the above program the **scanf()** statement returns the value equal to the number of variables correctly read. Here, it reads all the three variables values and hence v contains the value **3**. Which is printed in the last **printf( )** statement.

**Program 9** **Program to find the area of a triangle**

```
#include <stdio.h>
main( )
{
        float area, base, ht;
        printf ("Enter base and height \n");
        scanf ("%f %f", &base, &ht);
        area = 0.5*base *ht;
        printf ("Area = %f \n", area);
}
```

**Output**

```
Enter base and height
15.0 10.0
Area = 75.000000
```

**Explanation**

We have declared three variables **area, base** and **height** of type **float**. The first printf() prints the message as it is on the screen. The scanf() reads the values for base and height. After which the area is computed using the formula **0.5 * base * ht.** Finally last printf() prints the computed result.

**Program 10** **Suppose if we wish to enter the date in date format, then we can use '–' as delimiter.**

```
main( )
{
        int dd, mm, yy;
        printf ("\n Enter the date");
        scanf ("%d-%d-%d", &dd, &mm, &yy);
        printf ("Date is : %d-%d-%d", dd, mm, yy);
}
```

**Output**

```
Enter the date 20-07-2004
Date is : 20-07-2004
```

**Explanation**

While accepting the data, the user should type '-' delimiter, otherwise some garbage value will be stored into the variable. The above program shows how the ordinary characters can be used in the format string.

## 5.4.8 Field Width Specification for Inputting Integers

The scanf( ) function uses the field width to know how many columns on the screen should be used while reading a value from the keyboard. The syntax is same as printf().

**Example** **scanf ("%3d %3d %3d, &a, &b, &c);**

Here, %3d indicates that first three digits of the input number are significant and rest of the digits in that number are ignored. Such ignored numbers are input for the next data item or variable.

**Example**

```
void main( )
{
    int a, b, c;
    printf ("Enter the values for a, b & c \n")
    scanf ("%3d %3d %3d", &a, &b, &c);
    printf ("a = %d, b = %d, c = %d \n", a, b, c);
}
```

**Output**

```
1. If input is 25  35  45 then
   output is a = 25, b = 35, c = 45
2. If input is 111  222  333 then
   output is a = 111, b= 222, c = 333
3. If input is 1234  5678  90 then
   output is a = 123 b= 4 c = 567
```

**Explanation**

If the field width is not specified, then the input value will be stored in the corresponding variable until it gets any white space characters; otherwise the value will be stored till the width mentioned. If a number contains more digits than the specified width, then the rest of the digits are stored as a number to the next variable.

In the above case, 3rd input, the first variable 'a' stores 123 because its field width is specified as %3d, the digit '4' is stored as a number to the next variable 'b because after reading 'b' it encounters the white space and therefore only 4 is stored in 'b'. Finally next three digits are stored in 'c'. The remaining digits are ignored.

**Example**

Consider the following statements

```
scanf ("%2d %4d %d", &a, &b, &c);
printf ("a = %d, b = %d, c = %d", a, b, c);
```

(1) If we input 1234 567 89

Then the output is a = 12 b= 34 c = 567

(2) If we input 123 4567 89

Then the output is a = 12 b = 3 c = 4567

(3) If we input 12 34 5678

Then the output is a = 12 b = 34 c = 5678

## .4.9 Field Width Specification for Inputting Decimal Numbers

nlike Integer numbers, the field width of real numbers is not to be specified. The scanf() function ads the real numbers using %f format specifier for both notations - Decimal and Exponential.

**Example**

```
float x, y, z;
scanf("%f %f %f", &x, &y, &z);
if we input 3.142  42.21e+01  989.987 then
3.142 will be assigned to variable x, 42.21e+01 will be assigned to y and
989.987 will be assigned to z
printf("%f %f %f", x, y, z);
will print 3.142000, 422.100006, 989.987000
```

## 5.4.10 Field Width Specification for Inputting Character Strings

We have already seen how a single character can be read from the console using getchar(), getch() and getche() functions. The same can be achieved using scanf() function using %c format specifier. In addition, scanf() function can read strings containing more than one character using %s format specifier.

**Syntax:**

%Ws %Wc

Where

- W indicates total number of charcters to read from the console.
- s is the format specifier for string (sequence of characters).
- c is the format specifier for character.

When we use %Wc for reading a string, the system will wait until W^th^ Character is keyed in.

**Program 11 :** To read a single a character using scanf() and display the entered character using printf()

```
#include <stdio.h>
main( )
{
        char ch;
        printf(" Enter a Character : ");
        scanf("%c",&ch);
        printf("\n Entered Character is : %c", ch);
}
```

**Output**

```
Enter a Character : S
Entered Character is : S
```

**Explanation**

When a character is entered in the above program, the character itself is not stored. Instead, a numeric value(ASCII value) is stored. And when we displayed that value using "%c" format specifier, the entered character is displayed.

**Program 12 :** Printing ASCII value of the character

```
#include <stdio.h>
main( )
{
   char ch;
   printf(" Enter a Character : ");
   scanf("%c",&ch);
   printf("\n Entered Character is : %c ",ch);
   printf("\n ASCISSI value of the Character is : %d ",ch);
}
```

**Output**

```
Enter a Character : S
Entered Character is : S
ASCII value of the Character is : 83
```

**Explanation**

When %c format specifier is used for character variables, character is displayed.
When %d format specifier is used for character variables, ASCII value of the character is displayed

**Program 13** We can display a character if we know ASCII code of that character. This is shown by following example.

```
#include <stdio.h>
main( )
{
        int ch = 65;
        printf(" Character having ASCII value 65 is : %c",ch);
}
```

**Output**

```
Character having ASCII value 65 is : A
```

**Program 14** To read a string using scanf() and display using printf()

```
#include <stdio.h>
main( )
{
        char name[20];
        printf(" Enter a String : ");
        scanf("%s",&name);
        printf("\n Entered String is : %s ",name);
}
```

**Output 1**

```
Enter a String : Skyward
Entered String is : Skyward
```

**Output 2**

```
Enter a String : Skyward Books
Entered String is : Skyward
```

**Explanation**

In second output, even though Skyward Books was entered, only "Skyward" was stored in the name. It's because there was a space after Skyward. The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes blanks,tabs,carriage returns,form feeds and new lines.

Let us use format specification of %Wc to read the string until W<sup>th</sup> character is keyed in. To read the "Skyward Books", we will use "%13C" in the below example.

**Example** Using %Wc

```
#include <stdio.h>
main( )
{
        char name[20];
        printf(" Enter a String : ");
        scanf("%13c",&name);
        printf("\n Entered String is : %s ",name);
}
```

**Output**

```
Enter a String : Skyward Books
Entered String is : Skyward Books
```

**Explanation**

If we know the exact length of the string, then the above approach of using %Wc makes sense otherwise this is not a practical and not a generic solution to read a string with white spaces . A white space includes blanks,tabs,carriage returns,form feeds and new lines.

**Example** Using %Ws

```
#include <stdio.h>
main( )
{
        char name[20];
        printf(" Enter a String : ");
        scanf("%4s",name);
        printf("\n Entered String is : %s ",name);
}
```

**Output**

```
Enter a String : Bangalore
Entered String is : Bang
```

**Explanation**

Even though we typed 'Bangalore", scanf() read only 4 characters because we have specified field width as "%4s". If we use %20s, then 20 columns will be reserved and scanf can read maximum of 20 characters.

**How to use scanf to read Strings with Spaces ?**

We have seen that %s specifier cannot be used to read a string with white spaces. But this can be done with help of %[] specification. The format specifier "%[^\n]" tells to the compiler that read the characters until "\n" is not found.

**Example** To read string with blank spaces.

```
#include <stdio.h>
main( )
{
        char name[20];
        printf(" Enter a String : ");
        scanf("%[^\n]",name);
        printf("\n Entered String is : %s ",name);
}
```

**Output**

```
Enter a String :
Skyward Publishers Bangalore
Entered String is :
Skyward Publishers Bangalore
```

**Explanation**

See the output, now program is able to read complete string with blank spaces. The scanf() reads a string until new line character is encountered. Once we press the enter key it stops reading.

**How to read only specific characters using scanf() ?**

The scanf() function can read specific characters by using %[] specification.

| | | |
|---|---|---|
| **Example:** | 1. To read only lowercase letters | - Use %[a-z] |
| | 2. To read only upper case letter | - Use %[A-Z] |
| | 3. To read only upper case letters and digits | - Use %[A-Z,0-9] |
| | 4. To read characters of a,b, p,q,r,s | - Use %[abpqrs] |