Chapter-1

Introduction to the C Language

High Level Languages have been developed to facilitate easy programming of the computers by any ordinary person. One such HLL is the C Language. It is becoming so popular that the languages like BASIC, FORTRAN, and PASCAL are becoming obsolete. This is because C being a High Level language satisfies the varying requirements of programmers. It can be used for application program development as well as system program development.

1.History of C

The history of C starts with a language called BCPL (Beginners Combined Programming Language) developed by Martin Richards. In 1970, Mr. Ken Thompson, a System Engineer at AT & T Bell Laboratories, USA, wrote an earlier version of C language for the UNIX operating system. It is a modified version of the BCPL. Therefore to distinguish his language from BCPL, he called the language as B language, the first letter of BCPL. The B language was modified to a greater extent by Mr. Dennis Ritchie at Bell Labs. This modified version of B is called as the C language, the second letter of BCPL.

C was originally designed for UNIX operating system. But, later the whole UNIX operating system itself was almost rewritten in C language. C is a powerful, general purpose, procedure oriented, structured programming language.

2. About ANSI C Standard

For many years, the de facto standard for implementing the language has been the original C Reference Manual by Kernighan and Ritchie published in 1978. During these years, C has undergone many changes. Numerous different features and facilities have been developed and added. This has resulted in minor problems in portability. Consequently, the American National Standards Institute defined a standard for C, eliminating much uncertainty about the exact syntax of the language. This newcomer, called ANSI C, proclaims itself the standard version of the language. As such it will inevitably overtake, and eventually replace common C. ANSI C does incorporate a few improvements over the old common C. The main difference is in the grammar of the language. The form of function declarations has

been changed making them rather more like Pascal procedures. The most important ANSI additions are:

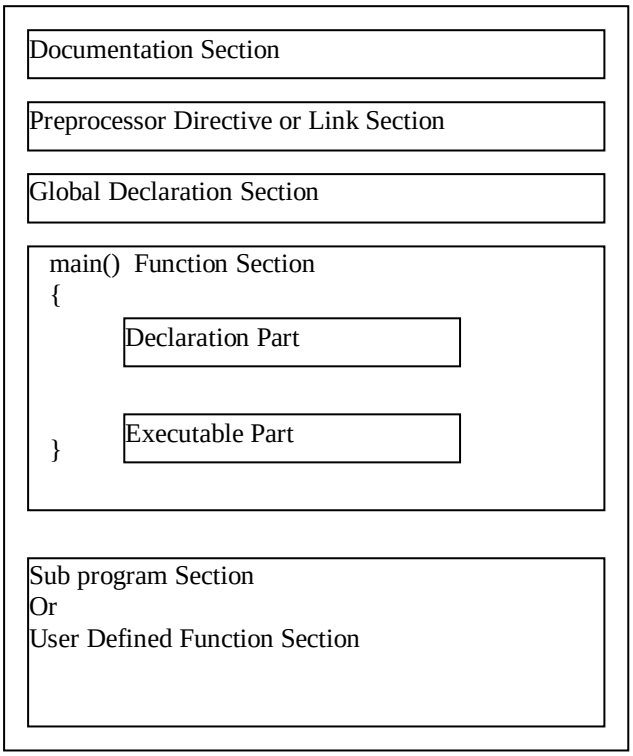
1. A new format for declaring and defining functions.
2. Facility for doing floating point computations.
3. Two new type qualifiers const and volatile and two new data type enum and void have been introduced.
4. The preprocessor includes some new macros.
5. A standard ANSI C library has been defined.

3. Important features of C Language

1. C is a system programming language which provides flexibility for writing compilers, operating systems, etc.
2. It can also be used for writing the application programs for scientific, engineering and business applications.
3. C is famous for its portability, meaning that program written in C for one computer can be easily loaded to another computer with little or no changes.
4. C supports variety of data types like integers, float point numbers, characters, etc.
5. C is a procedure oriented language which is most suited for structured programming practice.
6. It provides a rich set of built in functions
7. Programs written in C are found to execute faster than compared to other languages.

4. **Structure of the C Language**

The general structure of the C language includes the following sections



Example Program

```
/* Sample C Program */
main()
{
        int a,b;

        clrscr();

        printf("Enter two numbers");
        scanf("%d%d", &a,&b);

        sum = a + b;

        printf("Sum = %d", sum);
}
```

1. The Documentation Section : It consists of a set of comment lines giving the name of the program, the author and other details which the programmer would like to use later. The compiler ignores these comments when it translates the program into executable code. To identify the comment C uses two different formats:Block comments and Line comments

   Block comment: It is used when the comment will span several lines. It uses opening and closing comments. Each comment token is made of two characters. The opening token is /* and the closing token is */.Everything between the opening and closing comment token is ignored by the compiler.

   Line Comment: The line comment used two slashes (//) to identify a comment. This format does not require an end-of-comment token; the end of the line automatically ends the comment.

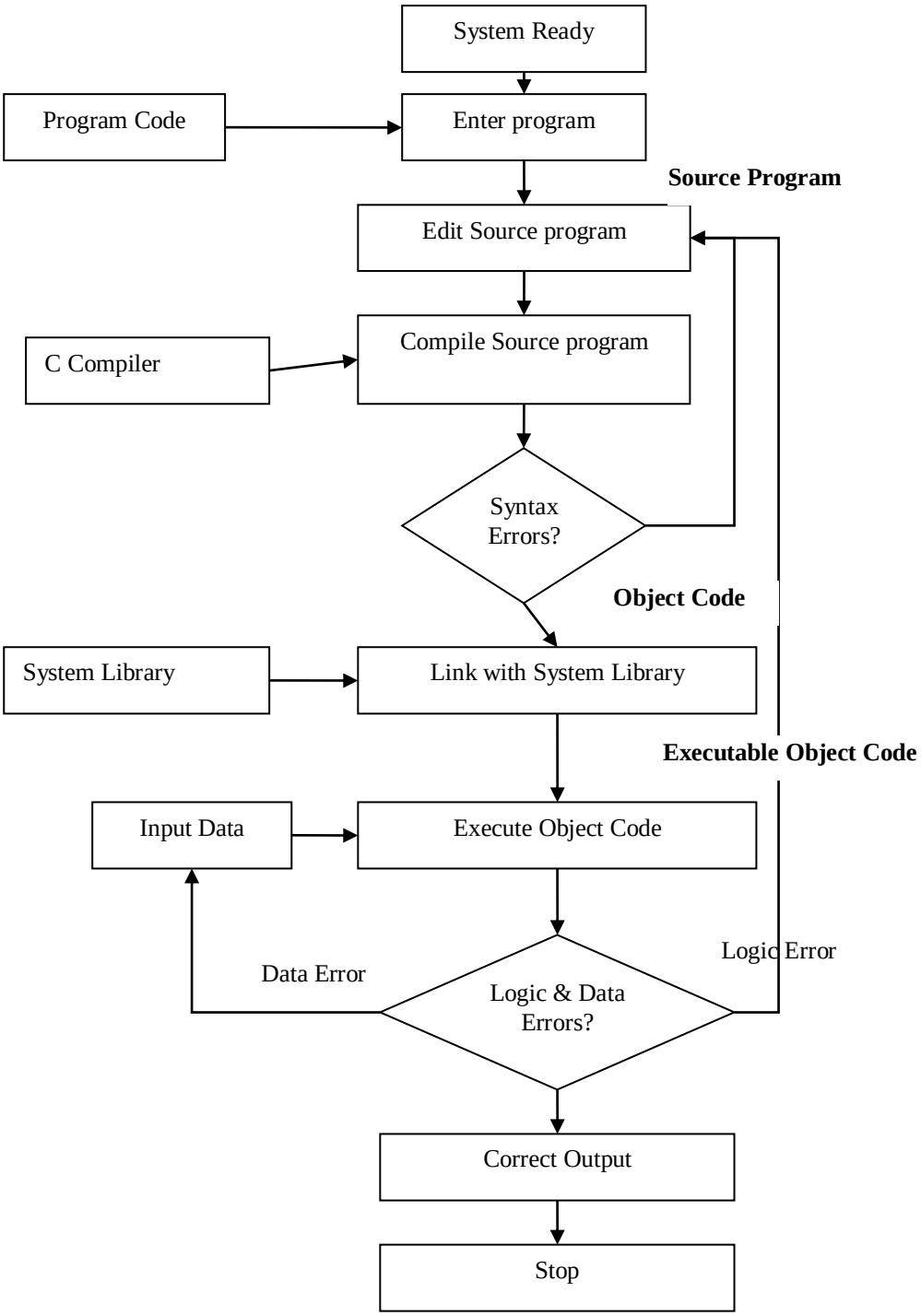
2. The Preprocessor directive or Link section : It provides instruction to the compiler to link some functions or do some processing prior to the execution of the program. It is also used to define symbolic constants of the program.
3. Global Declaration Section : There are some variables that are used in more than one function. Such variables are called global variables and are declared in this section that is outside of all other functions.
4. main() function section : Every C program must have one main() function. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is atleast one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration part and executable parts must end with a semicolon.
5. Sub program Section: It contains all the user defined functions that are called in the main () function. User defined functions are generally placed immediately after the main function, although they may appear in any order.

All sections except the main () function may be absent when they are not required in any C program.

5. Executing a C program

Executing a C program involves a series of following steps.

1. Creating a program
2. Compiling the program
3. Linking the program with functions that are needed from the C library.
4. Executing the program



In MS DOS :

For Compiling press Alt-F9.

For Compiling & Running the program press Ctr-F9.

For viewing the results press Alt-F5.

Important points to remember:

1. Every C program requires a main() function. Use of more than one main() is illegal. The place of main () is where the program execution begins.
2. The Execution of the function begins at the opening brace and ends at the closing brace.
3. C programs are written in lowercase letters. However uppercase letters may be used for symbolic names and constants.
4. All the words in a program line must be separated from each other by atleast one space or a tab, or a punctuation mark.
5. Every statement must end with a semicolon.
6. All variables must be declared for their type before they are used in the program.
7. Compiler directives such as define and include are special instructions to the compiler, so they do not end with a semicolon.
8. When braces are used in the program make sure that the opening brace has corresponding ending brace.
9. C is a free form language and therefore proper form of indentation of various sections would improve the legibility of the program.
10. A comment can be inserted almost anywhere a space can appear. Use of appropriate comments in proper places increases the readability of the program and helps in debugging an testing.

Identifiers:

Identifiers refer to the names of variables, functions and arrays. These are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers, but it is used as a link between two words in long identifiers.

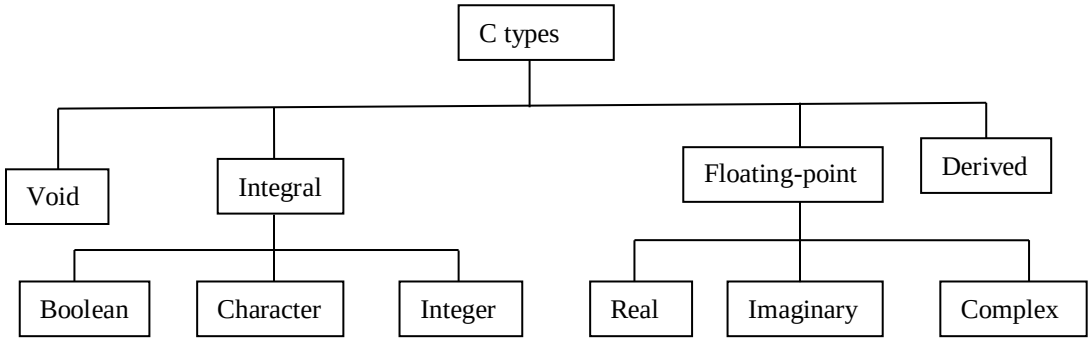
Rules for identifiers:

- First character must be an alphabet (or underscore).
- Must consist of only letters, digits or underscore.
- Only first 63 characters are significant.
- Cannot use a Keyword.
- Must not contain white space.

Valid Names	Invalid Names
a //Valid but poor style	$sum //$ is illegal
student_name	2names //First char digit
_aSystemName	Sum-salary //Contains hyphen
_Bool //Boolean System id	Stdnt Nmbr //Contains spaces
INT_MIN // System Defined Value	Int //Keyword

Types:

A type defines a set of values and a set of operations that can be applied on those values. The C language has defined a set of types that can be defined into 4 categories: Void, integral, floating-point and derived as following.



Void Types:

The void type has no values. It is used to represent the type of function. The type of function is said to be void when it does not return any value to the calling function.

Integral Type:

The C language has three integral types: Boolean, character and integer. Integral types cannot contain a fraction part; they are whole numbers.

Boolean:

A Boolean type can represent only two values: true or false. C used integers to represent the Boolean values: a nonzero number (Positive or negative) was used to represent true, and zero was used to represent false. The Boolean tyupe, which is referred to by the keyword bool, is stored in memory as 0 (false) or 1 (true).

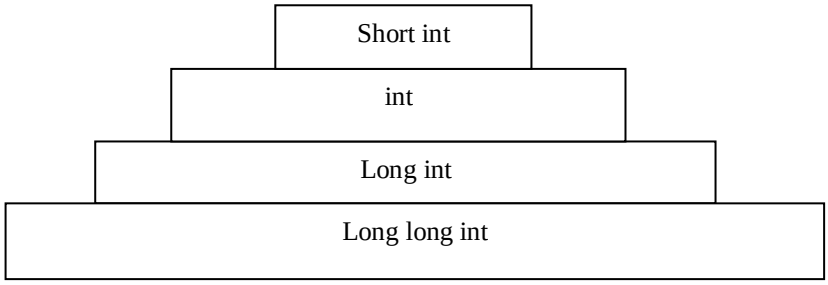
Character:

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifiers signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed chars have values from -128 to 127.

Integer:

Integers are whole numbers with range of values supported by a particular machine. Integers occupy one word of storage. The word sizes of machines vary the size of an integers i.e for 16 word length, the size of the integer value is limited to the range -32768 to +32767 (i.e -2^{15} to $+2^{15}-1$). A signed integer uses one bit for sign and 15 bits for the magnitude of the number. Similarly, a 32 bit word length can store an integer ranging from -2,147,483,648 to 2,147,483,648.

C has three classes of integer storage, namely short int, int and long int in both signed and unsigned forms as shown in figure:

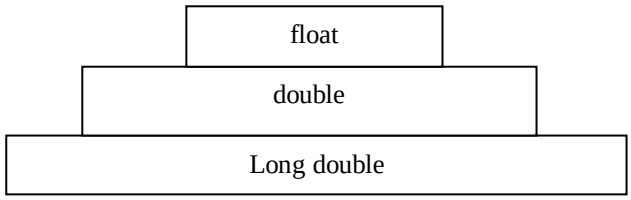


We declare long and unsigned integers to increase the range of values. The use of qualifier signed on integers is optional because the default declaration assumes a signed integer. Following table shows all combinations on a 16-bit machine.

Type	Size(bits)	Range
Char or Signed Char	8	-128 to 127
Unsigned Char	8	0 to 255
Int or Signed int	16	-32768 to 32767
Unsigned int	16	0 to 65535
Short int or Signed short int	8	-128 to 127
Unsigned short int	8	0 to 255
Long int or signed long int	32	-2147483648 to 2147483647
Unsigned long int	32	0 to 4294967295
Float	32	3.4 e-38 to 3.4 e+38
Double	64	1.7e-308 to 1.7e+308
Long Double	80	3.4 e-4932 to 3.4 e+4932
Long long int	8	-9,223,327,036,854,775,807 to -9,223,327,036,854,775,806

Floating Point types:

Float point (real) numbers are stored in 32 bits (on all 16 bit and 32 bit machine), with 6 digits of precision. It is defined by the keyword float. If the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. T extended the precision further, we may use long double which uses 18 bits. The relationship among them is represented by the following fig:



Variable:

Variables are named memory locations that have a type, such as integer or character which is inherited from their type.

Declaration of Variables:

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program. We must declare variables to the compilers, after designing variable names. Declaration does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will holds.

The declaration of variables must be done before they are used in the program.

Primary type declaration:

A variable can be used to store a value of any data type. That is, the name has nothing to do with its type. The syntax for declaration a variable is as follows:

Data-type v1, v2,.....,vn;

v1, v2,.....,vn are the n of variables. Variables are separated by commas. A declaration statement must end with a semicolon.

Ex: int count;
int number, total;
double ratio;

int and double are the keywords to represent integer type and real type data values respectively.

Data type	Keyword equivalent
Character	char
Unsigned character	unsigned char
signed character	signed char
signed short integer	signed short int(or short int or short)
signed long integer	signed long int (or long int or long)
unsigned integer	unsigned int (or unsigned)
unsigned short integer	unsigned short int (or unsigned short)
unsigned long integer	unsigned long int (or unsigned long)
floating point	float
double-precision floating point	double
extended double-precision floating point	long double

User-Defined Type Declaration:

C supports a feature known as "type definition" that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables.

syn: typedef type identifier;

Where type refers to an existing data type and "identifier" refers to the "new" name given to the data type. The existing data type may belong to any class of type, including the user-defined ones. The new type is 'new' only in name, but not the data type. Typedef cannot create a new type.

Ex: typedef int units;
typedef float marks;

Here, unit symbolizes int and marks symbolizes float. They can be later used to declare variables as follows:

Units b1, b2;
Marks m1, m2;

Here b1 and b2 are inclared as int variable and m1 and m2 are declared as floating point variables.

Another user-defined data type is enumerated data type.

enum identifier {v1, v2,....., vn};

The "identifier" is a user-defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces (enumeration constants). Then, we can declare variables to be of this 'new' type as below:

enum identifier v1, v2,...., vn;

The enumerated variables v1, v2,.....vn can only have one of the values v1, v2,.....,vn. We can assign the values to the variables as follows:

```
v1=value3;
v5=value1;
```

```
#include <stdio.h>
enum week{ sunday, monday, tuesday, wednesday, thursday, friday, saturday};
int main()
{
   enum week today;
   today=wednesday;
   printf("%d  day",today+1);
   return 0;
}
```

Output: 4 day

Here Wednesday is simply a name for an integer; it is not a string.

Ex: printf (“%s day",today);

The above code is wrong, since today is not a string.

Constants:

Constants in C refer to fixed values that do not change during the execution of a program. C supports different types of constants as shown below:

Constant Representation:

Boolean Constants:

A Boolean data type can take only two values. Those values are true and false. A Boolean value can have only one of the two values:0 (false) and 1 (true). For this, we have to include the Boolean Library, stdbool.h.

Single character Constants:

A single character constant or character constant is a single alphabet, a single digit or single special symbol enclosed within a pair of single quote marks.

Ex: ‘7’ ‘X’ ‘;’ ‘ ‘

Note that the character constant ‘7’ is not the same as the number 7.

Character constants have integer values known as ASCII values.

Ex: printf (“%d”,’a’) would print the number 97, the ASCII value of the letter a.

Similarly, the statement printf (“%c”,’97’) would print the letter ‘a’.

Note: - Each single character constant has an integer value that is determined by the

computer's particular character set.

Rules for Constructing Single Character constants

1. A single character constant or character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
2. The maximum length of a single character constant can be one character.
3. Each character constant has an integer value that is determined by the computer's particular character set.

Symbolic Name	ASCII Character
\a	Alarm or Beep
\f	Form Feed
\r	Carriage Return
\v	Vertical Tab
\'	Single Quote
\?	Question Mark
\b	Backspace
\n	New Line
\t	Tab (Horizontal)
\\	Backslash
\"	Double Quote
\0	Null

Integer Constants:

Integer constants are used to represent whole numbers. An integer constants can be specified in three types of integers namely, decimal, octal, or hexadecimal integer.

Decimal integers consist of a set of digits, 0 through 9, preceeded by an optional – or + sign.

Ex: 123 -321 0 654321 +8

Embedded spaces, commas& non-digit characters are not permitted between digits.

Ex: 15 750 20,000 $1000 are illegal.

An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0.

Ex: 037 0 0435 0551

Hexadecimal integers consist of a set of digits, preceeded by 0x or 0X. They may also include alphabets A through F or a through f which is represented by the numbers 10 through 15.

Ex: 0X2 0x9F 0Xbcd 0x

We rarely use octal and hexadecimal numbers in programming.

Real Constants:

Real constants are often known as floating point constants. Integer constants cannot represent quantities such as distances, heights, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts like 17.548. Such numbers are called real (or floating point) constants.

Ex: 0.0083 -0.75 435.36 +247.0

It is possible to omit digits before the decimal point, or digits after the decimal point.

i.e., 215. .95 -.71 +.5

A real numbers can also be expressed in exponential (or scientific) notation.

Ex: 215.65 can be expressed as 2.1565e2

Here e2 means multiply by 10^2.

i.e., mantissa e exponent

Here mantissa is either a real number expressed in decimal notation or an integer. The exponential is an integer number with an optional plus or minus sign. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase.

Ex: 0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1

Complex Constants:

Complex constants are coded as two parts, the real part and the imaginary part, separated by a plus sign. The real part is coded using the real format rules. The imaginary part is coded as a real number times (*) the imaginary constant (_Complex_I). For this, we have to include the complex library (complex.h).

String Constants:

It is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space.

Ex: “Hello!” “1987” “WELL DONE” “?.....!” “5+3” “X”

Ex: “X” ≠ ‘X’, since a single character string constant does not have an equivalent integer value while a character constant has an integer value.

Coding Constants:

There are three different ways that we code constants in our programs:

Literal constants, defined constants and memory constants.

Literal Constants:

A literal is an unnamed constant used to specify data. If, we know that the data cannot be changed, then we can simply code the data value itself in a statement.

Ex: a=b+5, Here 5 is the literal.

Defined Constants:

Another way to use the preprocessor command define. The defined constants are usually placed at the beginning of the program, although they are legal anywhere. This makes them easy to find and change.

Ex: # define Sales 0.90

Memory Constants:

Memory constants use a C type Qualifier, const, to indicate that the data cannot be changed.

Syn: const type identifier = value

Ex: const float PI=3.14;

INPUT/OUTPUT:

Formatted input:

Formatted input refers to an input data that has been arranged in a particular format.

Ex: 15.75 123 John

The first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in C using the scanf function. (scanf means scan formatted).

Scanf function:-

The function scanf() is used to read data into variables from the standard input, namely a keyboard. The general format is:

scanf ("Control string", var1,var2,……varn);

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2, ..., argn specify the address of locations where the data is stored. Control string and arguments are separated by commas.

- Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.
- Blanks, tabs, or newlines.

Blanks, tabs and newlines are ignored. The data type character indicates the type of data that is to be assigned to the variable associated with the corresponding argument. The field width specifier is optional.

Inputting Integer Numbers:

The field specification for reading a number is

% w sd

The percentage sign (%) indicates that a conversion specification follows, w is an integer number that specifies the field width of the number to be read and d, known as data type character, indicates that the number to be read is in integer mode.

Ex: scanf (“%2d %5d”,&m,&n);

Input: 89 39234

The value 89 is assigned to m and 39234 is assigned to n.

Input: 39234 89

The variable m will be assigned 39 and n will be assigned 234. The value 89 that is unread will be assigned to the first variable in the next scanf call. This kind of errors may be eliminated if we use the field specifications without the field width specifications.

Ex: scanf (“%d %d”,&m,&n);

Ex:
```
# include <stdio.h>
void main ()
{
        int a,b,c,x,y,z;
        int p,q,r;
        clrscr ();
        printf ("Enter three integer numbers\n");
        scanf ("%d %*d %d",&a,&b,&c);
        printf ("%d %d %d \n\n",a,b,c);
        printf ("Enter two 4-digit numbers\n");
        scanf ("%2d %4d",&x,&y);
        printf ("%d %d \n\n",x,y);
        printf ("Enter two integer\n");
        scanf ("%d %d",&a,&x);
        printf ("%d %d\n\n",a,x);
        printf ("Enter a nine digit number\n");
        scanf ("%3d %4d %3d",&p,&q,&r);
        printf ("%d %d %d\n\n",p,q,r);
        printf ("Enter two three digit number\n");
        scanf ("%d %d",&x,&y);
        printf ("%d %d",x,y);
        getch ();
}
```

Output:
```
Enter three integer numbers   1  2  3
1   3  -6107

 Enter two 4-digit numbers     6789   4321
67       89
Enter two integers       44       66
```

```
4321    44
Enter a nine-digit number       123456789
66      1234    567
Enter two three-digit numbers    123  456
89      123
```

Inputting Real Numbers:

Unlike integer number, the field width of real numbers is not to be specified and therefore scanf reads real numbers using the simple specification %f for both the notations, namely, decimal point notation and exponential notation.

Ex: scanf (“%f %f %f”,&x,&y,&z);

Input: 475.89 43.21E-1 678

Will assign the value 475.89 to x, 4.321 to y, and 678.0 to z.The input field specifications may be separated by any arbitrary blank spaces.

Ex:
```
# include <stdio.h>
void main ()
{
    float x,y;
    double p,q;
    printf ("Enter the values of x and y:: ");
    scanf ("%f %e",&x,&y);
    printf ("\n");
    printf ("x = %f\ny = %f\n\n",x,y);
    printf ("Values of p and q :: ");
    scanf("%lf %lf",&p,&q);
    printf ("\n\np = %.12lf\np = %.12e",p,q);
    getch ();
}
```

Output:
```
Values of x and y:      12.3456         17.4e-2
x = 12.345600
y = 0.175000
Values of p and q :     4.142857142857      18.5678901234567890
p = 4.142857142857
q = 1.856789012346e+001
```

Inputting Character Strings:

A scanf function can input strings containing more than one character. Following are the specifications for reading character strings:

%ws or %wc

The use of %wc for reading a string, the system will wait until the w^{th} character is keyed in.

Ex:
```
void main ()
{
    int n;
    char x[15],y[15],z[15];
    printf (“Enter serial no and name-1”);
    scanf (“%d %15c”,&n,&x);
    printf (“%d %15s”,n,x);
    printf (“Enter serial no and name-2”);
    scanf (“%d %s”,&n,&y);
    printf (“%d %15s”,n,y);
    printf (“Enter serial no and name-3”);
    scanf (“%d %15s”,&n,&z);
    printf (“%d %15s”,n,z);
}
```

Output:
```
Enter serial no and name-1  1  123456789012345
                            1  123456789012345
Enter serial no and name-2  2  New York
                            2              New
Enter serial no and name-3  3  London
                            3          London
```

Some versions of scanf support the following conversion specifications for strings:

%[characters]

%[^characters]

The specification %[characters] means that only the characters specified within the brackets are permissible in the input string. If the input string contains any other character, the string will be terminated at the first encounter of such a character. The specification %[^characters] does exactly the reverse i.e., the reading of the string will be terminated at the encounter of one of these characters.

Ex:
```
# include <stdio.h>
void main ()
{
    char a[80];
    printf ("Enter the address\n");
    scanf ("%[a-z]",a);
```

```
        printf ("%-80s\n\n",a);
        getch ();
    }
```

Output:
```
Enter the address
Newdelhi        110002
Newdelhi
```

Ex:
```
# include <stdio.h>
void main ()
{
        char a[80];
        printf ("Enter the address\n");
        scanf ("%[^\n]",a);
        printf ("%-80s\n\n",a);
        getch ();
}
```

Output:
```
Enter the address
Newdelhi        110     002
Newdelhi        110     002
```

Formatted Output:

Printf function can be used for printing captions and numerical results. The general form of printf statement is:

Printf (“Control String”,arg1, arg2, ..., argn);

Control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence characters such as \n, \t, and \b.

The control string indicates how many arguments follow and what their types are. The arguments arg1, arg2, ..., argn are the variables.

A simple format specification has the following form:

% w . p type-specifier

Output of integers:

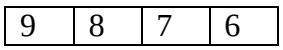
Format specification for printing an integer number is :

% w d

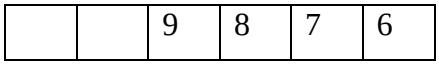
Where w specifies the minimum field width for the output. If a number is greater than the specified field width, it will be printed in full, overriding the minimum specification. d specifies that the value to be printed is an integer. The number is written right-justified in the given field width.

Ex: Format Output

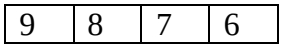
Printf (“%d”,9876)



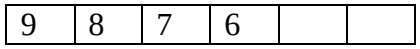
Printf (“%6d”,9876)



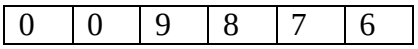
Printf (“%2d”,9876)



Printf (“%-6d”,9876)



Printf (“%06d”,9876)



Output of real numbers:

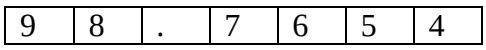
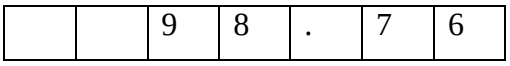
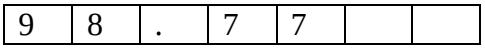
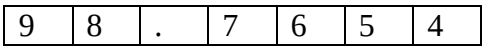
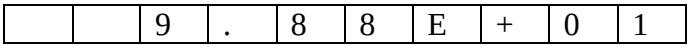
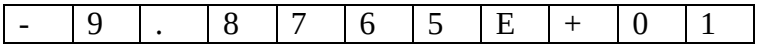
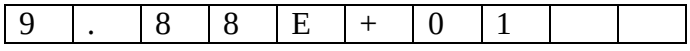
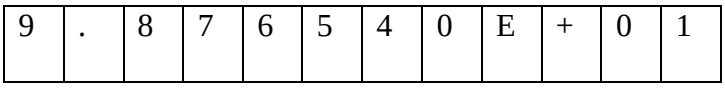
The output of real numbers may be displayed in the decimal notation using the following format specification.

% w . p f

Where w indicates the minimum number of positions that are to be used for the display of the value and the integer p indicates the number of digits to be displayed after the decimal point (precision).

- A minus sign, which specifies left adjustment of the converted argument.
- A number that specifies the minimum field width. The converted argument will be printed in a field at least this wide. If necessary it will be padded on the left or right, to make up the field width.
- A period, which separates the field width from the precision.
- A number, the precision, that specifies the maximum number of characters to printed from a string, or the number of digits after the decimal point of a floating point value, or the minimum number of digits for an integer.

Ex: y = 98.7654

Format	Output
Printf (“%7.4f”,y)	
Printf (“%7.2f”,y)	
Printf (“%-7.2d”,y)	
Printf (“%f”,y)	
Printf (“%10.2e”,y)	
Printf(“%11.4e”,y)	
Printf(“%-10.2e”,y)	
Printf(“%e”,y)	

Some systems also support a special field specification character that lets the user define the field size at run time.

Printf (“%*.*f”,width,precision,number);

Ex: printf (“%*.*f”,7,2,number);

is equivalent to printf (“%7.2f”,7,2,number);

Printing of a Single character:

A single character can be displayed in a desired position using the format:

% w c

The character will be displayed right-justified in the field of w columns. We can make the display left-justified by placing a minus sign before the integer w.The default value for w is 1

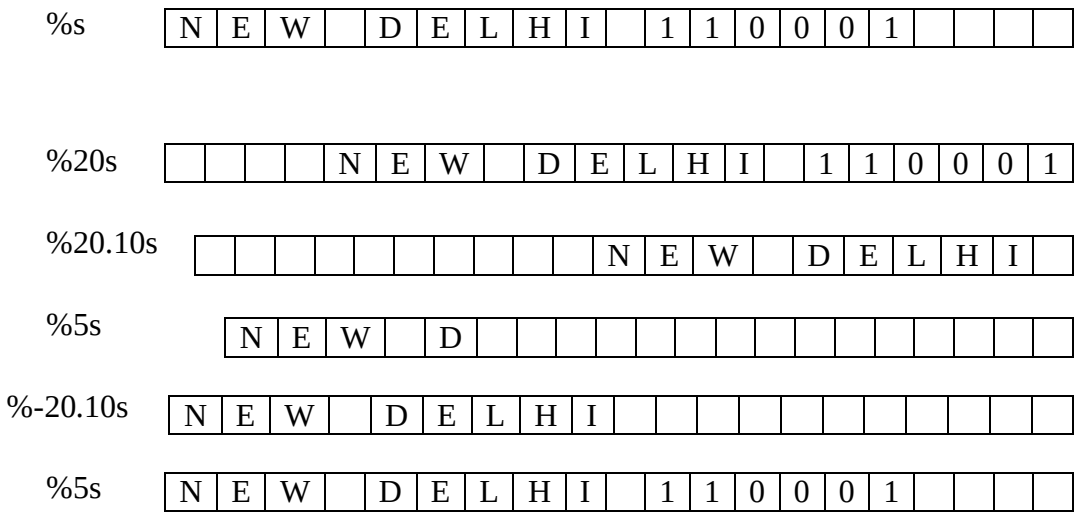
Printing of strings:

The output of strings may be displayed using the following format specification.

% w . p s

Ex: “NEW DELHI 110001”

Specifiation Output



Conversion characters are shown in the table below:

CHARACTER	INPUT DATA
d	decimal integer
i	integer. The integer may be in octal (leading 0) or hexadecimal
o	octal integer(with or without leading zero);
u	unsigned decimal integer;
x	hexadecimal integer (with or without leading ox or ox)
c	character
s	character string (not quoted)
e,f,g	floating-point number with optional sign, optional decimal point and optional exponent;
%	literal %; no assignment is made.

Mixed Data Output:

We can use mixed data types in one printf statement.

Ex: printf (“%d %f %s %c”,a,b,c,d);