

## **Programming for Problem Solving**

**UNIT - I**

**L- 09**

**Introduction to Algorithms and Programming Languages:** Basics of algorithms, Flow charts, Generations of programming languages.

**Introduction to C:** Structure of a C program; pre-processor statement, inline comments, Variable declaration statement, Executable statement; C Tokens: C Character Set, Identifiers and Keywords, Type Qualifiers and Type Modifiers, Variables and Constants, Punctuations, and Operators.

### **Problem Solving:**

It is a process of finding an efficient solution for the given problem. As computer has no self or natural intelligence like humans we need to guide computer how to solve a problem by providing a set of well defined steps/instructions.

### **Steps to solve the Problem:**

1. Understand and analyse the problem.
2. Design the solution by using appropriate programming tool (Algorithm, Pseudo code, Flowchart).
3. Develop a code for the designed solution using programming language.
4. Run and test the solution using for correctness and efficiency.
5. Document the solution.

### **Introduction to Algorithm:**

An algorithm is written in a natural language. It is a step – by – step procedure to solve a specific problem.

**Example:** Find the sum of two numbers.

Algorithm: Find the sum of two numbers

- Step 1: Start  
Step 2: Declare variables num1, num2 and sum.  
Step 3: Assign values num1=10, num2=20, sum=0.  
Step 4: Set sum=num1+num2.  
Step 5: Print sum.  
Step 6: Stop.

### **Characteristics of good algorithm:**

Input: Accepts zero (or) more inputs.

**Definite:** Steps are precisely defined and unambiguous (if we provide wrong instructions computer will perform wrong).

**Output:** Good algorithm generates one (or) more outputs.

**Finite:** An algorithm should be finite. That is an algorithm should terminate after a finite number of steps.

**Efficient:** It should be efficient in terms of memory usage and time.

### **Key features of Algorithm:**

An algorithm has a finite number of steps and some steps may involve decision making, repetition, and sequence.

**Sequence:** Here each step of the algorithm is executed in the specific order.

Example: Sum of two numbers.

Algorithm: Sum of two numbers

- Step 1: Start
- Step 2: Declare variables num1, num2 and sum.
- Step 3: Assign values num1=10, num2=20, sum=0.
- Step 4: Set sum=num1+num2.
- Step 5: Print sum.
- Step 6: Stop.

### **Flowchart**

- Flowchart is a graphical representation to solve a problem in step by step manner.
- Flowchart is used to indicate flow of information.
- Instead of descriptive steps (Algorithm), it uses pictorial representation for every step.

### **Limitations:**

- ✓ Flow charts are difficult to modify (redrawing)
- ✓ Translation of flow chart into computer program is always not easy.

### **Advantages:**

- Communication: Flowcharts are better way of communicating the logic of a program.

- Effective analysis: With the help of flowchart, problem can be analysed in more effective way therefore reducing cost and wastage of time.

## **Flowchart Symbols**

### **Terminal**

- *The beginning or ending of a program*
- *It is single directional i.e. only one flow line(either incoming line or outgoing line) can be connected with terminal symbol.*



### **Input/Output**

- *The parallelogram is used to represent input and output operation.*
- *It is having two flow lines, i.e., incoming and outgoing lines.*



### **Process**

- *The rectangle symbol is used to represent mathematical calculations / process.*
- *It is having two flow lines, i.e., incoming and outgoing lines.*



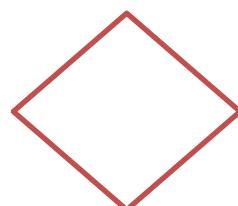
### **Flowline**

- *The arrow symbol is used to establish a logical flow between symbols.*

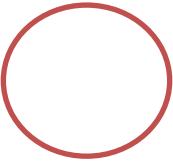


### **Decision**

- *The rhombus symbol is used to check the condition i.e., decision.*
- *It is having single flow line for input and two flow lines for output.*



### **Connector**



- *The circle is used to connect two portions of flowcharts in different pages.*
- *It is having single flow line (either incoming line or outgoing line)*

**Let us discuss with an example:**

**Your father wants to pay salary to his workers. He need to calculate the ‘Total Pay’ every time.**

**Task:** Pay calculator

**Input:** No. of hours worked ( $n$ ) and Hourly pay rate ( $h$ ).

**Process:** ???

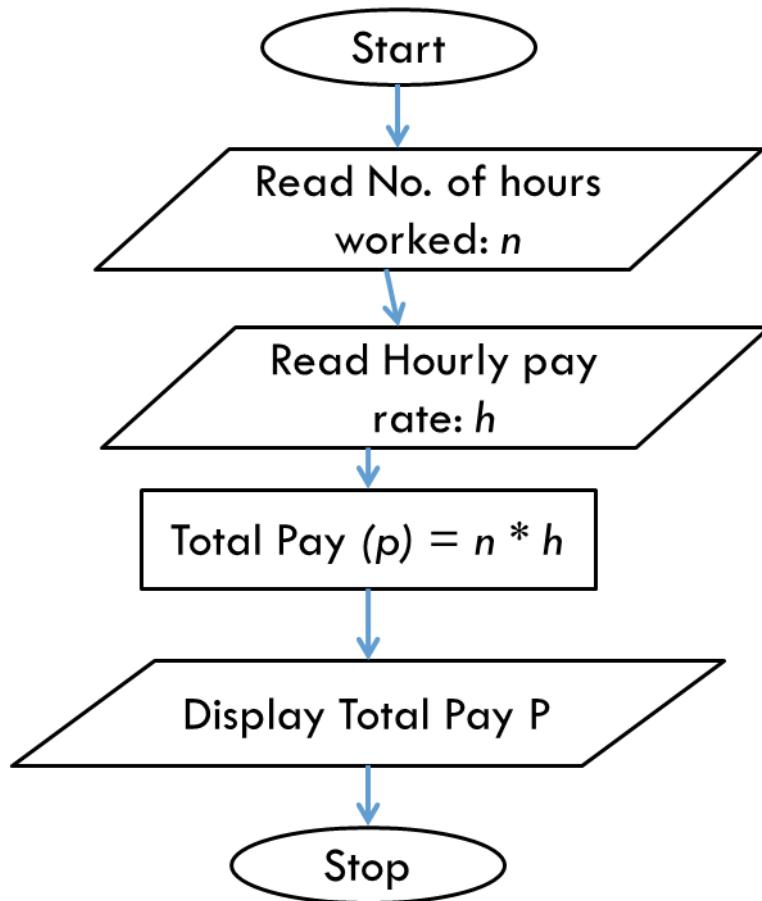
$$\text{Total Pay } (p) = n * h$$

**Output:** Total Pay ( $p$ ).

**Pseudocode:**

1. Display “No. of hours worked:”
2. Input  $n$
3. Display “Hourly pay rate:”
4. Input  $h$
5. Calculate Total Pay
  1.  $\text{Total Pay } (p) = n * h$
6. Display “Total Pay ”
7. Output  $p$

### **Flow chart for Pay Calculator**



### **Generations of Programming Languages**

There are five generation of Programming languages. They are:

#### **First Generation Languages:**

These are low-level languages like machine language.

#### **Second Generation Languages:**

These are low-level assembly languages used in kernels and hardware drives.

#### **Third Generation Languages:**

These are high-level languages like C, C++, Java, Visual Basic and JavaScript.

#### **Fourth Generation Languages:**

These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting.

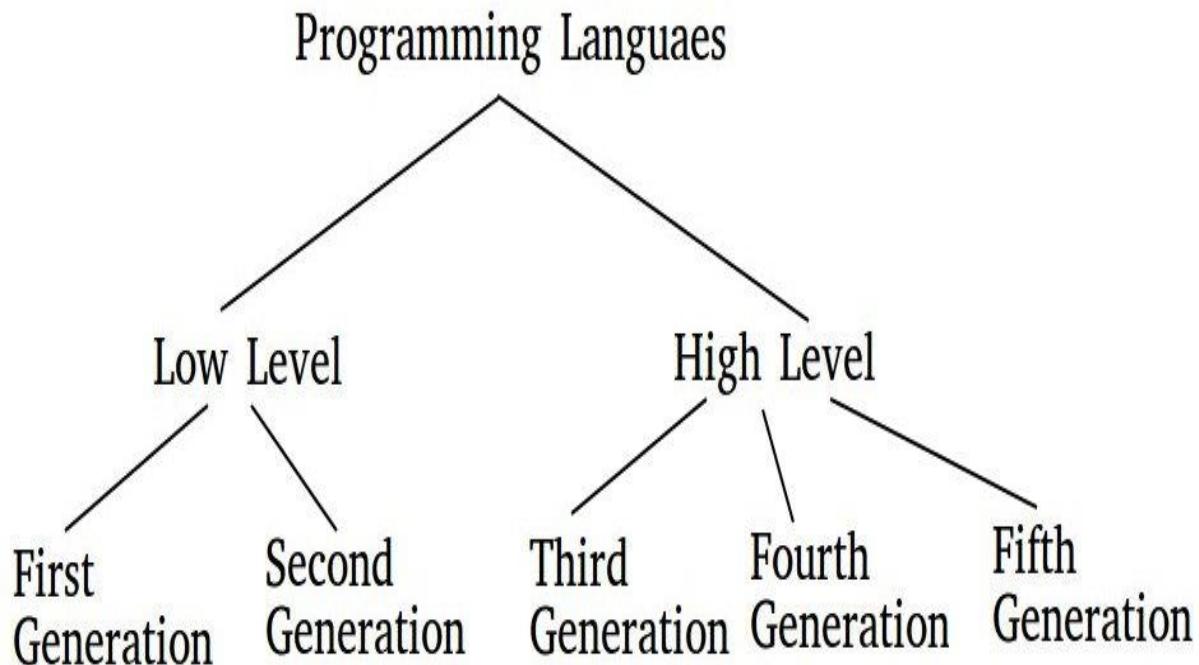
Example of these languages include Perl, Python, Ruby, SQL, MatLab(MatrixLaboratory).

### **Fifth Generation Languages:**

These are the programming languages that have visual tools to develop a program.

Examples of fifth generation language include Mercury, OPS5, and Prolog.

The first two generations are called low level languages. The next three generations are called high level languages.



### **Introduction to C:**

C programming language is most popular programming language. C was created in 1972 by Dennis Ritchie at the Bell Labs in USA as a part of UNIX operating system. C was also used to develop some parts of this operating system.

Most of the operating systems like Linux, Windows™, and Mac™ are either developed in C language.

Possibly why C seems so popular is because it is reliable, simple and easy to use.

There are several reasons why we need to learn C even it is a part of C++, C#, and JAVA:

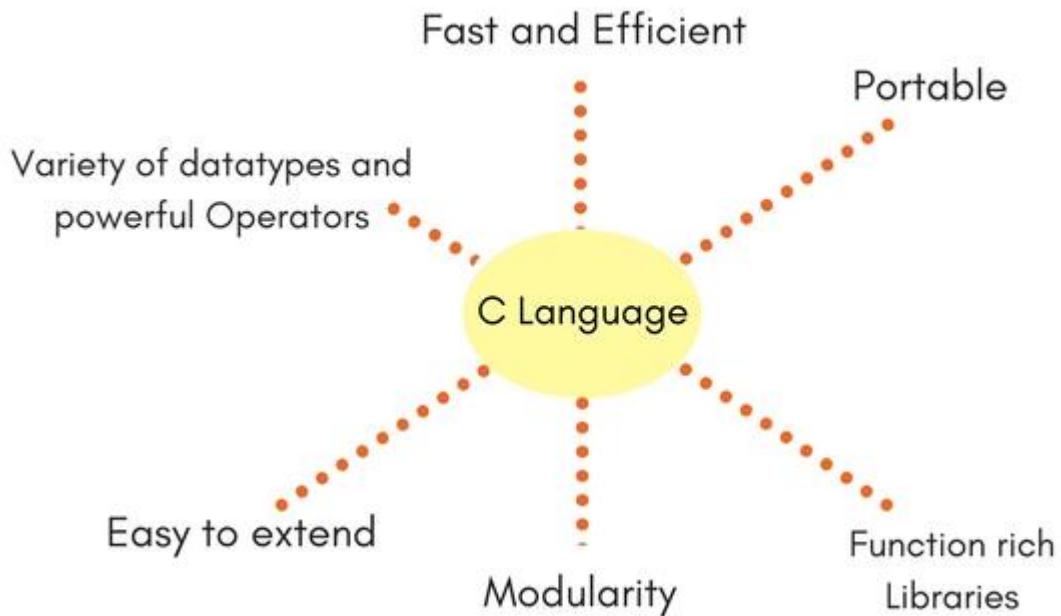
- I believe that nobody can learn C++ or Java directly. This is because while learning these languages we have things like classes, objects, inheritance, polymorphism, templates, exception handling, etc. to deal with apart from knowing the actual language elements.
- Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C.
- Mobile devices like cellular phones and palmtops are becoming increasingly popular. Also, common consumer devices like microwave oven, washing machines and digital cameras are getting smarter by the day. This smartness comes from a microprocessor, an operating system and a program embedded in this devices. These programs not only have to run fast but also have to work in limited amount of memory. No wonder that such programs are written in C. With these constraints on time and space, C is the language of choice while building such operating systems and programs.

I hope that these are very convincing reasons why one should adopt C as the first and the very important step in your quest for learning programming languages.

### **Features of C:**

- It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.
- The C compiler combines the capabilities of an assembly language with features of a high-level language.
- Programs Written in C are efficient and fast. This is due to its variety of data type and powerful operators.
- C is highly portable this means that programs once written can be run on another machines with little or no modification.
- Another important feature of C program is its ability to extend itself.

- A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
- C language is the most widely used language in operating systems and embedded system development today.



### **Structure of C Program:**

A C program involves the following sections:

- Documentations (Documentation Section)
- Pre-processor Statements (Link Section)
- Global Declarations (Definition Section)
- The `main()` function
  - Local Declarations
  - Program Statements & Expressions
- User Defined Functions

## **Documentation Section / Comments –**

They are two types of comments:

- Single – line comment.
- Multi – line comment.

**Single – line Comment:** It starts with two forward slashes (i.e //) and is automatically terminated with the end – of the line. The first line of our program is a single line comment.

**Multi – line Comment:** It starts with /\* and terminates with \*/. It is used when multiple lines of text are to be commented.

## **Pre-processor Directive Section**

The following points must be remembered while writing a pre-processor directives:

- The pre-processor directive should always start with hash symbol (#).
- The # symbol should be the first non-white space character in a line.
- It will terminated with the new line character not with semicolon.
- Pre-processor directives are executed before the compiler compiles the source code.

### **Some important pre-processor directives:**

#define      Substitutes a pre-processor Macro (constant)

#include     Inserts a particular header file from another file.

#ifdef Returns true if the macro is defined.

#ifndef     Returns true if the macro is not defined.

#if       Tests if a compile time condition is true.

#else      The alternative of #if.

#elif     #if and #else in one statement.

#endif     Ends pre-processor conditional statement.

#error     Prints error message on console output device.

## **Global Declaration Section-**

- In this section global variables and user defined functions are declared before main().
- These functions can be accessed by all used defined functions including main().
- If a user defined function or a variable is not declared in this section it will not be accessed by the remaining functions.

## **Function Section –**

This section is mandatory. This section have one or more functions. A function named main() is always required. Every function consists of two parts:

- Header of the Function.
- Body of the Function.

### **Header of the Function:**

- The general form of the header of the function is:
  - [return-type] function\_name([argument\_list])
- The terms enclosed within square brackets are optional.

### **Body of the Function:**

- The body of a function consists of a set of statements enclosed with in braces {}. And these statements are of two types:
  - Non – Executable Statements (Ex – Declaration Statements)
  - Executable Statements (Ex – Function call statements like printf(..))
- It is possible that no statement present within the braces. In such case the program produces no output on execution.
- If any statements present in the function non-executable statements should present prior to the executable statements.

## **A Simple program that prints Hello World!**

```
// Comment: First C program
#include<stdio.h>
main()
{
    printf("Hello World! \n");
}
```

Output:  
Hello World!

## **Compile and Execute a C Program:**

Let us see how to save the source code in a file, how to compile and how to run it.

- Open a text editor and add the above mentioned code.
- Save the file as hello.c.
- Open command prompt and go to the directory where we have saved the file.
- Type cc file\_name.c and press enter to compile our code.
- If there are no errors in our code, the command prompt will take you to the next line and would generate a.out executable file.
- Now type ./a.out to execute our program.
- Finally we will get output.

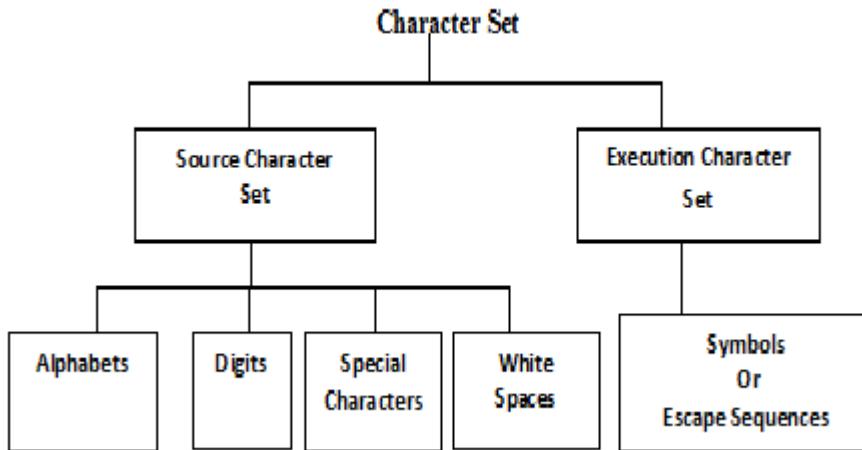
```
$cc hello.c
$./a.out
Hello world!
```

## **C Character Set:**

A character set defines the valid characters that can be used in a source program (or) interpreted when a program is running.

The set of characters that can be used to write a source program is called ‘Source Character set’. And the set of characters that can be available when the program is being executed is called an ‘Execution Character set’.

It is possible that the source character set is different from the execution character set, but in most cases they are identical.



## Source Character Set

This character set is used to construct statements in a source program.

Alphabets	A to Z and a to z
Digits	0 to 9
Special characters	+,-,*,!,<,>,?, @, #. The others are listed below in detail
Whitespaces	Blank space, horizontal tab, vertical tab, new line, form feed

## Special Characters are listed below

Symbol	Name	Symbol	Name
~	Tilde	-	Minus sign
!	Exclamation Mark	=	Equal to sign
#	Number sign	{	Left flower brace
\$	Dollar sign	}	right flower brace
%	Percentage sign	[	Left bracket
^	Caret	Right bracket	
&	Ampersand	:	Colon
*	Asterisk	“	Quotation mark
(	Left Parenthesis	<	Opening angle bracket (less than sign)
)	Right Parenthesis	;	Semicolon
_	Underscore	>	Closing angle bracket (greater than sign)
+	Plus sign	?	Question mark
	Vertical bar	,	Comma

\	Backslash	.	Period (dot)
'	Apostrophe	/	Slash
@	At symbol		

## Execution Character Set

Execution characters are used at the time of execution. They are invisible and cannot be printed or displayed correctly. They are always represented by a backslash(\) followed by a character. They are also known as non graphic characters, symbols or escape sequences. Some of them are given below

## Symbols

Symbol	Name	Symbol	Name
'\b'	Blank space	'\f'	Form feed
'\n'	Newline	'\r'	Carriage return
'\t'	Horizontal tab	'\v'	Vertical tab

## Identifiers

An Identifier refers to the name of an object. It can be a variable name, a function name, a typedef name, a macro name, or a structure name, or a union name.

The syntactic rules to write an identifier name in C are as follows:

1. It should have letters, digits or underscores.
2. The first character of an identifier must be a letter not digit.
3. No special characters can be used (, ).
4. Keywords are not used for identifiers.
5. The maximum number of letters in an identifier depends on compiler.

Ex: Student\_Name, StudentName, Student123..

## Keywords –

Keyword is a reserved word that has a particular meaning in the programming language. The meaning of the keyword is predefined. A keyword cannot be used as an identifier name in C language. There are 32 keywords available in C. The following table shows the list of keywords:

Auto	double	int	struct
Break	else	long	switch

Auto	double	int	struct
Case	enum	register	typedef
Char	extern	return	union
Const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
Do	if	static	while

## **Variables –**

Variable is a name given to Memory location in which data is stored. Variable acts as value, which changes during the execution of program. Variable may take different value at different times during execution.

### **The general format of any variable declaration**

**datatype v1, v2, v3, ..... vn**

where v1, v2, v3 are variable names. Variables are separated by commas. A declaration statement must end with a semicolon.

**Eg:**                    int sum;

                          double average, mean;

## **Variable Assignment**

It is another format of variable declaration. The general form is

**data-type variable name [=value];**

Here data type refers to the type of value used in the C program. Variable name is a valid name.

**Eg:**                    int a; (or)    int a=10;

Here if the value of a is assigned 10 the value remains the same throughout the program. If no value is specified then the value keeps on changing.

## **Simple Variable Assignment**

To assign a single value to a variable '=' equality operator is used. The syntax is

**variable-name = value;**

**Eg:**            X=10; y=30;

## **Compound Variable Assignment**

**variable-name = expression;**

This type of variable declaration can be used together with combination of characters, numbers or an expression. Here '=' equality operator is used.

**Eg:**            Y=x+5;        Z= ((x+1) (y-2)\*x)

## **Declaring a variable as constant**

Variable is declared as constant by using the keyword or qualifier '**const**' before the variable name. This can be done at the time of initialization.

**Eg: const** int class\_size = 40;

## **Volatile Variable**

A variable is volatile if the value gets changed at any time by some of the external sources.

**Eg: volatile** int num;

When we declare a variable as volatile the compiler will examine the value of the variable each time it is encountered to see if an external factor has changed the value.

## **Constants**

A Constant is an entity whose value remains the same throughout the execution of the program. These fixed values are also called as **Literals**. They are classified into three types:

1. Literal Constants.
2. Qualified Constants.
3. Symbolic Constants.

## 1. Literal Constants

They are just called as Literals denotes a fixed value, which may be an integer, floating point number, character or a string.

### Integer Literal Constant

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Following are other examples of various types of integer literals –

```
85      /* decimal */
0213    /* octal */
0x4b    /* hexadecimal */
30      /* int */
30u     /* unsigned int */
30l     /* long */
30ul    /* unsigned long */
```

### Floating Point Literal Constant

It can be written in a Fractional form or Exponential Form. The rules for writing floating point literal constants are:

1. A fractional floating point literal constant must have at least one digit.
2. It should have a decimal point.
3. It can be either positive or negative.
4. No special characters or blank spaces are allowed.

Here are some examples of floating-point literals –

```
3.14159  /* Legal */
314159E-5L /* Legal */
210f      /* Illegal: no decimal or exponent */
.e55      /* Illegal: missing integer or fraction */
```

### Character Literal Constant

It can have one or at most two characters enclosed within single quotes e.g. 'A', 'a', '\n' etc. These are classified into two types:

1. Printable Character Literal Constant.
2. Non – Printable Character Literal Constant.

- 1. Printable Character Literal Constant:** All characters of source character set except quotation mark, backslash, and new line character when enclosed within a single quotes form a Printable Character Literal Constant. Ex: 'A', '2'..
- 2. Non – Printable Character Literal Constant:** These are represented with the help of 'Escape Sequences'. An escape sequence consists of backward slash (\) followed by a character and both enclosed within a single quote. Ex: '\n', '\t'..

### List of Escape Sequences

S.NO	Escape Sequence	Character Value	Action on Device	Output
1	\'	Single quotation mark	Prints '	
2	\"	Double quotation mark	Prints "	
3	\?	Question Mark	Prints ?	
4	\\\	Backslash character(\)	Prints \	
5	\a	Alert	Generates beep sound	
6	\b	Back space	Moves the cursor one position to the left of its current position.	
7	\f	Form Feed	Moves the cursor to the beginning of the next page	
8	\n	New line	Moves the cursor to the beginning of the next line.	
9	\r	Carriage Return	Moves the cursor to the beginning of the current line.	
10	\t	Horizontal Tab	Moves the cursor to the next horizontal tab stop	

**String Literal Constant:** It consists of a sequence of characters. Here each constant is implicitly terminated by a null character. So the length of the string constant is +1 than the actual length.

**Qualified Constants:** These are created by using 'const' qualifier.

Ex: const int a=10;

Constants are not able to modified, but they are able to access.

**Symbolic Constants:** These are created with the help of define pre-processor directive.

Ex: #define pi 3.14

## **Punctuations –**

The punctuation and special characters in the C character set have various uses, from organizing program text to defining the tasks that the compiler or the compiled program carries out. They do not specify an operation to be performed.

## **Syntax**

( ) [ ] { } \* , : = ; ... #

These characters have special meanings in C. # occurs only in pre-processor directives.

## **Type qualifiers and Type Modifiers –**

The declaration statement can optionally have type qualifiers or type modifiers or both.

### **Type qualifiers –**

C – type qualifiers: The keywords which are used to modify the properties of a variable are called type qualifiers.

#### **TYPES OF C TYPE QUALIFIERS:**

There are two types of qualifiers available in C language. They are,

1. const
2. volatile

### **Type Modifiers –**

A type modifier alter the meaning of the base data type to yield a new type. There are four types of modifiers in C they are

- signed
- unsigned
- short
- long

## **Operators**

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Conditional Operators
6. Special Operators
7. Bitwise operators
8. Increment and decrement operators
9. Unary operators
10. Equality operators