

**Paper – 104 Computer Programming & Programming Methodology
(CPPM)**

Subject Code -104

Subject title - Computer Programming & Programming Methodology

Review/Revision Year: June 2020

UNIT-1: Introduction

1.1 Concepts of Programming Language

1.1.1 Introduction of Source Code, Object Code and executable code

1.1.2 Algorithm and Flowchart

1.1.3 Concepts of Structured Programming Language

1.2 Concepts of Editor, Interpreter and Compiler

1.2.1 Introduction of C program body structure

1.2.2 Character Set, concepts of variables and constants

1.2.3 Identifiers, literals, Key words

1.2.4 Data types (signed and unsigned) (Numeric : int, short int, long, float, double) , (Character type: char, string) and void.

1.2.5 Concepts of source code, object code and executable code.

UNIT-2: Input/Output Statements and Operators:

2.1 Input/Output statements:

2.1.1 Concepts of Header files (STDIO, CONIO)

2.1.1.1 Concepts of pre-compiler directives.

2.1.1.2 Use of #include and #define

2.2 Input/Output Statements:

2.2.1 Input statements : scanf(), getc(), getch(), gets(), getchar()

2.2.2 Output Statements: printf(), putc(), puts(), putchar()

2.2.3 Type specifiers (formatting strings) : %d, %ld, %f, %c, %s, %lf

2.3 Operators :

2.3.1 Arithmetic operators (+, -, *, /, %, ++, --,)

2.3.2 Logical Operators (&&, ||, !)

2.3.3 Relational Operators (>, <, ==, >=, <=, !=)

2.3.4 Bit-wise operators (&, |, ^, <<, >>)

2.3.5 Assignment operators (=, +=, -=, *=, /=, %=)

2.3.6 Ternary Operator and use of sizeof() function.

2.4 Important Built-in functions:

2.4.1 Use of <string.h> : (strlen, strcmp, strcpy, strcat, strcmp)

2.4.2 Use of <math.h> : (abs(), floor(), round(), ceil(), sqrt(), exp(), log(), sin(), cos(), tan(), pow() and trunc())

UNIT-3: Decision Making statements :

3.1 if statements :

3.1.1 simple if statements

3.1.2 if...else statements

3.1.3 if...else if...else statements

3.1.4 Nested if statements.

3.2 Switch..case statements

3.2.1 Use of break and default

3.2.2 Difference between switch and if statements.

UNIT-4: Iterative statements :

4.1 Use of goto statement for iteration

4.2 while loop

4.3 do..while loop

4.4 for loop

4.5 Nested while, do..while and for loops

4.6 Jumping statement: (break and continue)

UNIT-5: Concepts of Arrays and pointer

5.1 Concepts of Single-dimensional Array

5.1.1 Numeric single dimensional Array

5.1.2 Numeric single dimensional array operations:

5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)

5.1.2.2 Searching element from array (Linear Search)

5.1.3 Character Single dimensional Array

5.1.3.1 Character Single dimensional array operations:

5.1.3.2 Use of \0, \n and \t

5.2 Pointers:

5.2.1 Concepts of Pointers

5.2.2 Declaring and initializing int, float, char and void pointers

5.2.3 Pointer to single dimensional numeric array.

Reference Books-

1. Programming in C, Balaguruswami – TMH
2. C: How to Program, Deitel & Deitel - PHI
3. C Programming Language, Kernigham & Ritchie - TMH
4. Programming in C, Stephan Kochan - CBS
5. Mastering Turbo C, Kelly & Bootle - BPB
6. C Language Programming – Byron Gottfried - TMH
7. Let us C, Yashwant Kanetkar - BPB Publication
8. Magnifying C, Arpita Gopal - PHI
9. Problem Solving with C, Somashekara - PHI
10. Programming in C, Pradip Dey & Manas Ghosh – Oxford

UNIT-1: Introduction

1.1 Concepts of Programming Language

1.1.1 Introduction of Source Code, Object Code and executable code

1.1.2 Algorithm and Flowchart

1.1.3 Concepts of Structured Programming Language

1.2 Concepts of Editor, Interpreter and Compiler

1.2.1 Introduction of C program body structure

1.2.2 Character Set, concepts of variables and constants

1.2.3 Identifiers, literals, Key words

1.2.4 Data types (signed and unsigned) (Numeric : int, short int, long, float, double) , (Character type: char, string) and void.

1.2.5 Concepts of source code, object code and executable code.

1.1 Concepts of Programming Language

- A **program** is a set of instructions that help computer to perform tasks.
- The languages used to write a program or set of instructions called as "**Programming languages**".
- Some example of programming language are C, C++, Java, Python

1.1.1 Introduction of Source Code, Object Code and executable code

- Source code– A code created by programmer using programming language known as source code/program. E.g. .c files
- Object code– Source code given as input to the compiler and produce object code. An object code is also known as binary code/machine code. E.g. .obj file
- **Executable code**- object code links with linker and system library and generate executable file known as executable code. E.g .exe files

Source code → compiler → **object code** → linker (runtime library files) → **executable code**

1.1.2 Algorithm and Flowchart

Algorithm- An algorithm is a step-by-step procedure (process) to Solve given problem.

Algorithms have a definite beginning and a definite end, and a finite number of

Steps. An algorithm produces output depending on the basis of given input, and several algorithms combine to perform complex tasks such as writing a computer program.

- **Features of Algorithm**

- It should be simple.
- It should be finite number of steps.
- It should be clear with no uncertainty (doubt).
- It have the capability to handle some unexpected situations which may arise during the solution of a problem.

Example 1. 1

Write an Algorithm to addition of two numbers.

Step 1: Start

Step 2: Take two numbers

Step 3: Addition of two numbers

Step 4: Display result.

Step 5: Stop

Example 1. 2

Write an Algorithm to calculate the Area of Circle

Step 1: Start

Step 2: Take the value of radius

Step 3: $\text{area} = 3.14 * \text{radius} * \text{radius}$

Step 4: print area

Step 5: Stop

Above example the steps are execute sequentially but sometimes the steps are skipped and execute based on the condition to making decision.

Example 1. 3

Write an algorithm to print whether a number is positive or negative

Step 1: Start

Step 2: Take a number

Step 3: if number > zero so print “Positive number” and go to step 5 otherwise go to next step 4.

Step 4: Print “Negative number and go to step 5

Step 5: Stop

Sometimes we want to repeat one or more statements more the once. It called looping.

Example 1 4

Write an algorithm to print 1 to 10 numbers

Step 1: Start

Step 2: Initialize the value of variable I =1

Step 3: Check condition I is ≤ 10 , if condition is true than print I and go to Step 4 otherwise go to step 5

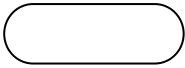





Step 4: Increment the value of I ($I=I+1$) and go to step 3

Step 5: Stop

Flowchart

- Flowchart is a graphical way to represent the algorithm.
- Flowchart is very helpful to solve the complicated programming problem.
- Once the flowchart is drawn, it has become easy to write a program.

The following symbols used to draw flowchart.

	Start and stop
	Calculation
	Input and Output
	Decision
	Flow line
	Connector

- **Benefits of Flow chart**

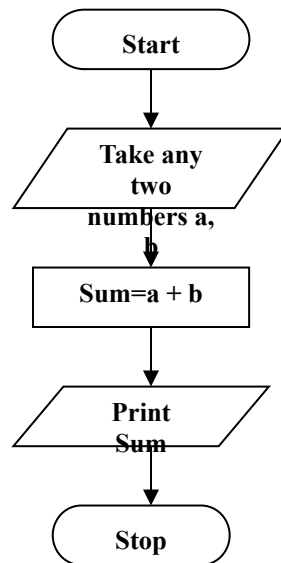
- Problem easily understood.
- Proper Program documentation.
- Error tracking is simple.

- **Limitation of Flow chart**

- Complex & detailed flowcharts are sometimes difficult to draw.
- Sometimes difficult to add the information in flowchart.

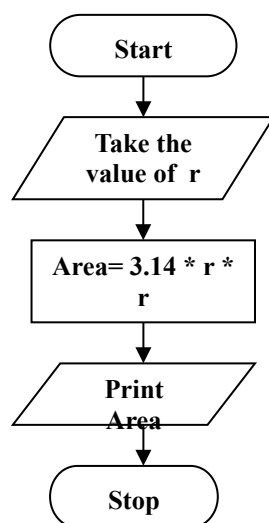
Example 1. 5

Draw the flowchart to addition of two numbers.



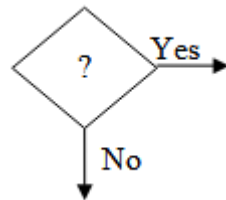
Example 1. 6

Draw the flowchart to calculate the Area of Circle



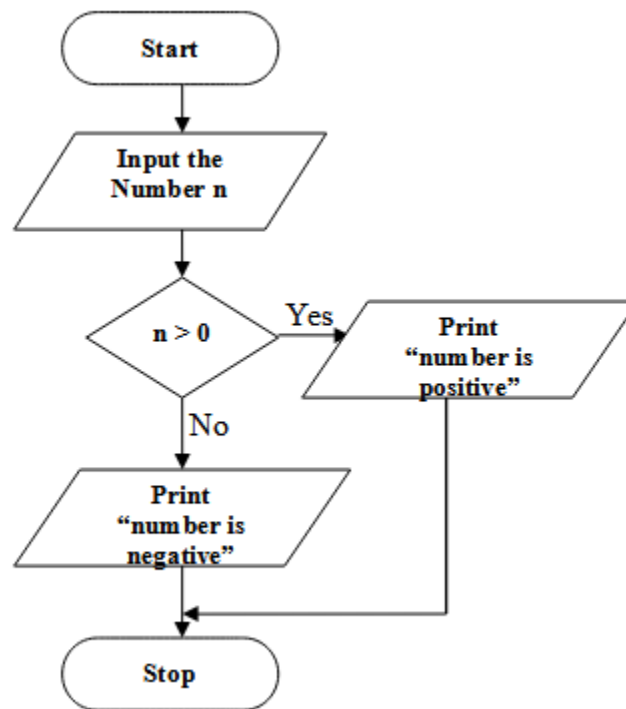
- **Branching**

- Branching refers the statement may or may not be executed based on condition.
- The condition has written inside the decision box.
- If the condition is true then yes branch have been executed otherwise no is execute.



Example 1. 7

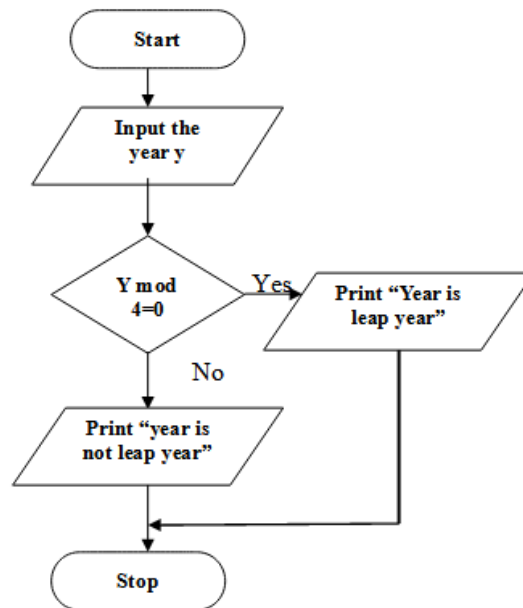
Draw the flowchart to print the number is Positive or Negative.



We can use relational operators ($>$, $<$, $=$, $>=$, $<=$, $<>$) for decision-making.

Example 1. 8

Draw the flowchart to find whether the given year is leap year or not.



Example 1. 9 Write an algorithm and draw flowchart to check no is positive, negative and zero

Step 1 start

Step 2 take a number Num

Step 3 if Num > 0 then

 Display Num is positive

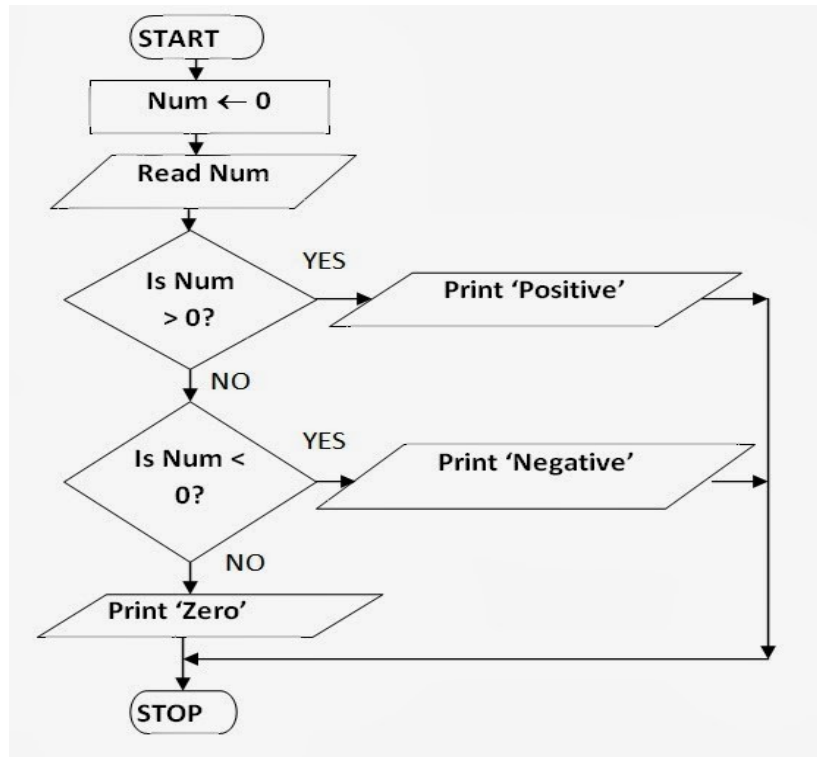
Else Num < 0 then

 Display Num is negative

Else

 Display Num is zero

Step 4 stop



Example 1. 10 find largest no from three no

1-start

2- Take 3 numbers a,b,c

3- Check condition

 If $a > b$ and $a > c$

 Display "a is largest"

 Otherwise check $b > a$ and $b > c$

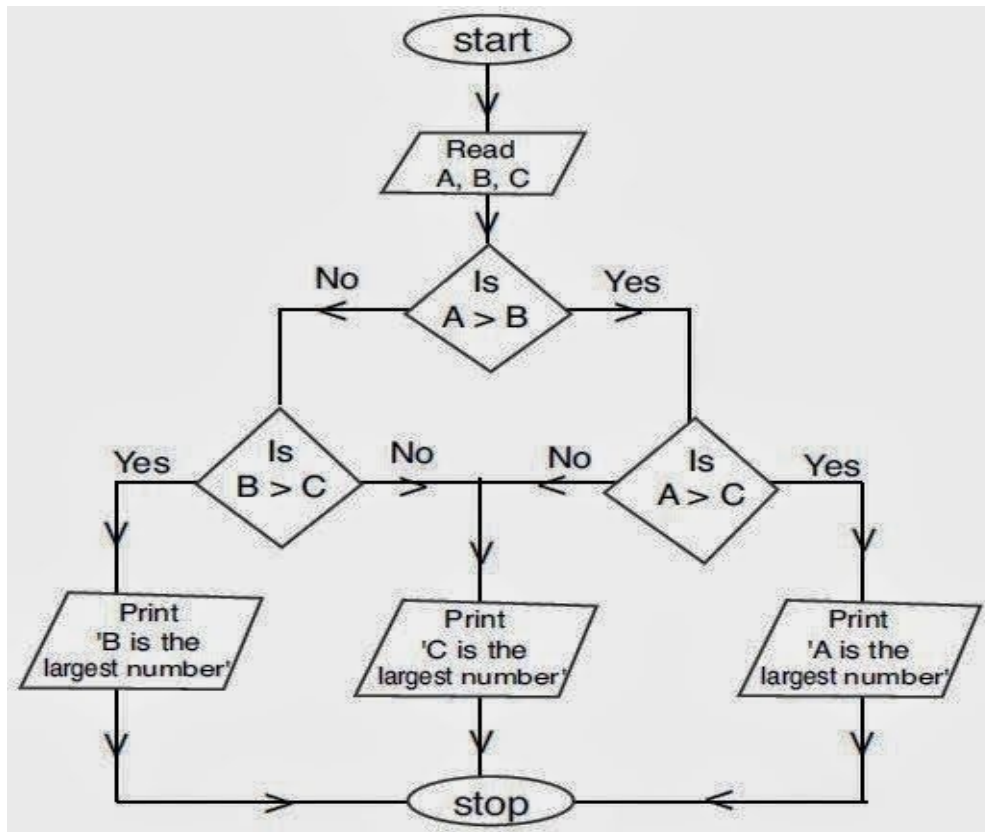
 Display "b is largest"

 Otherwise

 Display "c is largest"

4-stop

Draw the flowchart to find largest from three nos.

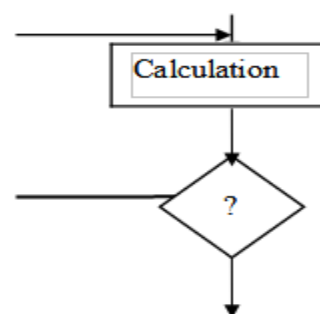
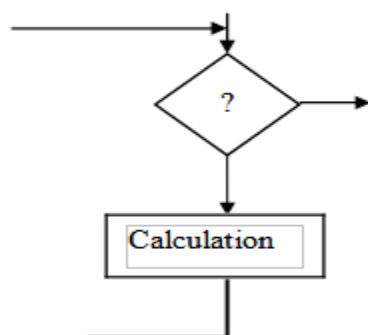


- **Looping**

- Looping refers to repeated use of one or more steps.
- There are two types of loops.

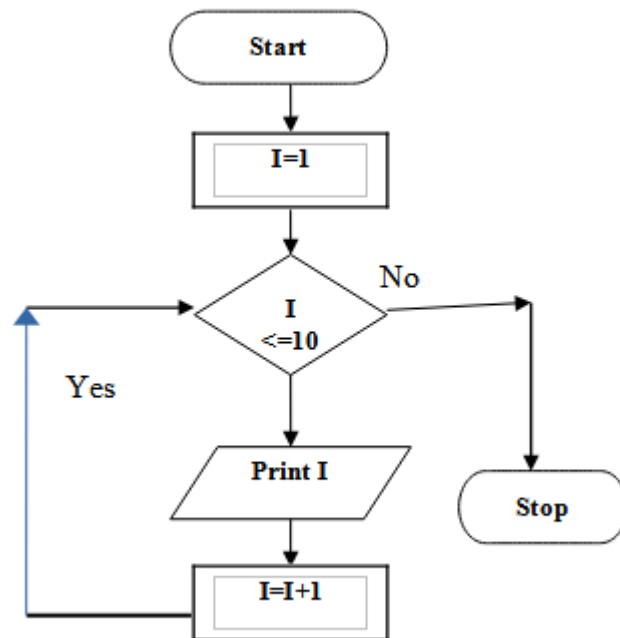
(1) fixed loop and (2) variable loop.

- In *fixed loop* set of statements are executed fixed no of times until the condition is true and
- In *variable loop* statement are executed until the specified condition is met.
- Loops can jump forward or back ward.
- Loops are defined as follow:



Example 1. 10

Draw the flowchart to print 1 to 10 numbers.



Example 1.11 find sum of N number $s=1+2+3+...+n$

1-start

2-take value of n(input n)

3- Initialize the value of $I = 1$ and $s=0$

4-Check condition $I \leq n$

 If condition is true than

$s=s+i$

 goto step 5

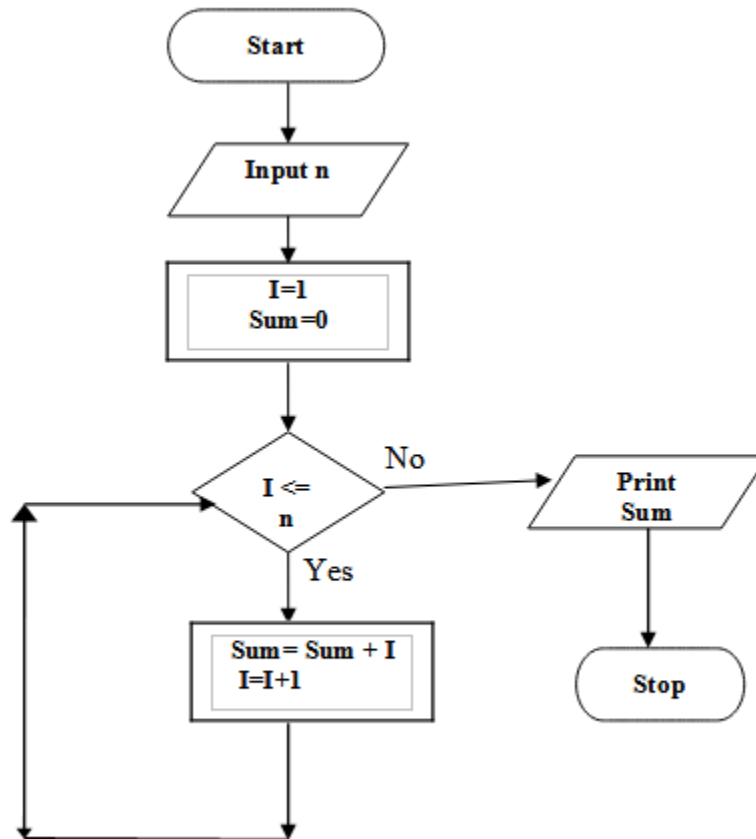
 otherwise goto step 6

5- Increment the value of I ($I=I+1$) and go to step 4

6- display s

7-Stop

Draw the flowchart to calculate sum of n^{th} numbers ($1+2+3\dots n$)



1.1.3 Concepts of Structured Programming Language

C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility. The program that solves the entire problem is a collection of such functions.

Here is an example of Matrix addition program, which is divided into these sub-procedures - input matrix, display matrix, add matrix, save result matrix to file.

Advantages

- C structured programming is simple and easy to understand and implement.
- It is well suited for small size implementation. However, this is not restricted. A good design can extend it to large size implementation.
- Programmers do not require knowing complex design concepts to start a new program.

Disadvantages

- Polymorphism and inheritance are not available.
- Complex design and full object oriented design have not be implemented.
- Programmers generally prefer object oriented programming language to structured programming language when implementing a complex gaming applications or front-end business applications.

1.2 Concepts of Editor, Interpreter and Compiler

- 1.2.1 Introduction of C program body structure

A C program may contain one or more sections. They are illustrated in figure 2.1 below.

1. **Documentation section** : The documentation section 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.

Two types of comments

Single line(//) and multiline (/*.....*/)

2. **Link section** : The link section provides instructions to the compiler to link functions from the system library. E.g. #include <stdio.h>
3. **Definition section** : The definition section defines all symbolic constants. e.g. #define a 10
4. **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 the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.
5. **main () function section** : Every C program must have one main function section. This section contains two parts; declaration part and executable part
6. **Declaration part** : The declaration part declares all the variables used in the executable part.
7. **Executable part** : There is at least 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 and executable part end with a semicolon.

8. **Subprogram section** : The subprogram section 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 section, except the **main ()** function section may be absent when they are not required.

Example 2.1

//Write program to print your name

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

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
printf("\n My Name is : Ved patel");
```

```
getch();
```

```
}
```

Output

My Name is: Ved patel

- 1.2.2 Character Set, concepts of variables and constants

Character set are the set of alphabets, letters and some special characters that are valid in C language.

Alphabets:

Uppercase: A B C X Y Z

Lowercase: a b c x y z

Digits:

0 1 2 3 4 5 6 8 9

Special Characters:

Special Characters in C language

, < > . _ () ; \$: % [] # ?

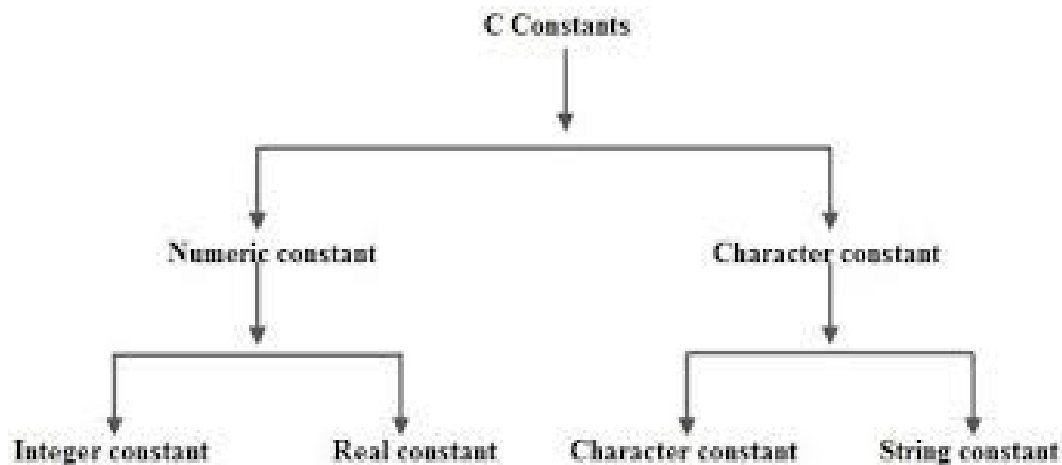
' & { } " ^ ! * / | - \ ~ +

White space Characters:

Blank space, new line, horizontal tab, carriage return and form feed

Constant and variable –needs and definition

Constants cannot be changed during the execution of a program. In C, constants can be classified as:



Numeric constant

Integer constants

Integer constants are the numeric constants without any fractional part.

It may be positive or negative.

Space and comma(,) and non digit character are not allowed.

For example 123, +12,-12

There are three types of integer constants in C language: decimal constant (base 10), octal constant (base 8) and hexadecimal constant (base 16) .

Decimal digits: 0 1 2 3 4 5 6 7 8 9

Octal digits: 0 1 2 3 4 5 6 7

Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

For example:

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

Real (Floating-point) constants

Floating point are the numeric constants with any fractional part.

It may be positive or negative.

Space and comma(,) and non digit character are not allowed.

For example 12.3, +1.2, -1.2

Character constants

Character constants

Character constants are the constant which use single quotation around characters. For example: 'a', 'l', 'm', 'F' etc.

String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"           //string constant
""               //null string constant
"  "             //string constant of six white space
"x"              //string constant having single character.
"Earth is round\n" //prints string with newline
```

Escape Sequences

Escape Sequences	Character
------------------	-----------

\b	Backspace
\f	Form feed
\n	Newline
\r	Return

\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

Variables –needs and definition

Variable are used to store data.

Value of the variable may be change during program execution.

Rules for writing variable name

First character must be letter and remaining character may be letter, numbers or underscore.

Uppercase and lower case letters are different

Space, special characters accept underscore are not allow.

The variable name should not be keyword.

It should be short and meaningful.

Variable must be declared before they are used in the program.

Variable can be assigned by using assignment operator.

Syntax variable = value

Example A=10

It also possible to assign value at the time of variable declaration

Syntax datatype variable = value

Example int A = 10

C allows initialization of more than one variable in one statement using multiple assignment operators.

Example void main()

```
{  
    int a,b,c;  
    a=b=c=10;  
}
```

- 1.2.3 Identifiers, literals, Key words

Identifiers

Identifiers are names given to variables, functions, structures etc. For example:

```
int A;
```

Here, A is a identifier which denotes a variable of type integer.

Keywords

Keywords are the reserved words. Each keyword has fixed meaning and that cannot be changed by user. For example:

```
int A;
```

Here, int is a keyword that indicates, 'A' is of type integer.

C programming is case sensitive; all keywords must be written in lowercase. Here is the list of all keywords predefined by C.

Keywords in C Language			
auto	double	int	Struct
break	else	long	Switch
case	enum	register	Typedef

char	extern	return	Union
continue	for	signed	Void
do	if	static	While
default	goto	sizeof	volatile
const	float	short	unsigned

- 1.2.4 Data types (signed and unsigned) (Numeric : int, short int, long, float,double) , (Character type: char, string) and void.

1. Fundamental Data Types

- o Integer types
- o Floating Type
- o Character types

2. Derived Data Types

- o Arrays
- o Pointers
- o Structures
- o Enumeration

Syntax for declaration of a variable

```
data_type variable_name;
```

Integer data types

Keyword int is used for declaring the variable with integer type. For example:

```
int A;
```

Here, A is a variable of type integer.

Floating types

Variables of floating types can hold real values (numbers) such as: 2.34, -9.382 etc. Keywords either float or double is used for declaring floating type variable.

For example:

```
float B;  
double C;
```

Here, both A and C are floating type variables.

When the number declared by the float is not sufficient then type double is used.

If double is not sufficient then long double is used.

Character types

Keyword char is used for declaring the variable of character type. For example:

```
char n='h';  
char name[10]="ved"
```

Here, n and name are variables of type character. For understanding following table shows size and range of different data type.

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

- 1.2.5 Concepts of source code, object code and executable code.

Source code— The program written in high level language (HLL) is known as source code/program. Source code is given as input to the compiler.

Object code— the program written in HLL and converter to machine level language (MLL) is known as object code /program. An object code is also known as binary code/machine code.

Translator – it is required to convert source program (HLL) into object program (MLL). Compiler and interpreter both is translator. Main difference between compiler and interpreter explain in below.

No	Compiler	Interpreter
1	Compiler Takes Entire program as input	Interpreter Takes Single instruction as input .
2	Intermediate Object Code is Generated	No Intermediate Object Code is Generated
3	Conditional Control Statements are Executes faster	Conditional Control Statements are Executes slower
4	Memory Requirement : More (Since Object Code is Generated)	Memory Requirement is Less
5	Program need not be compiled every time	Every time higher level program is converted into lower level program
6	Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted (if any)
7	Example : C Compiler	Example : BASIC