

## 1. Write about History of C-Language?

**History of C:** C was an offspring of the “Basic Combined Programming Language (BCPL) called B, developed in the 1960’s at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named C.

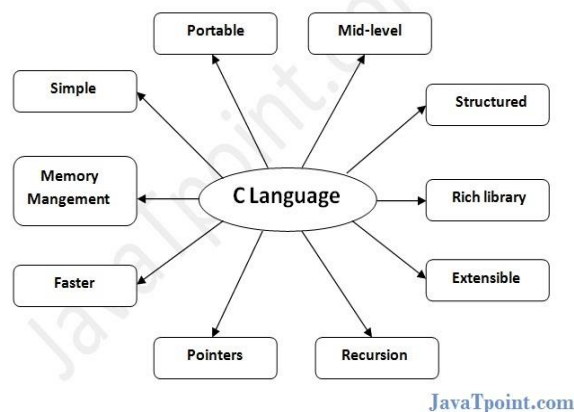
**Importance of C:** The increasing popularity of c is probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used to write any complex program, the compiler combines the capabilities of any assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages .In fact many of the C compiler available in the market are written in c.

Programs written in c are efficient and fast. This is due to its variety of data types and powerful operators. It is many times faster than BASIC. For example, a program to increment a variable from 0 to 15000 takes about one second in c while it takes more than 50 seconds in an interpreter BASIC.

There are only 32 keywords and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs. C is highly portable. This means that C programs written for one computer can be run on another computer with little or no modification.

C language is well suited for structured programming, thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. Another important feature of c is its ability to extend itself.

## 2. Explain about Features of C-Language?



C is the widely used language. It provides a lot of **features** that are given below:

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

**1) Simple:**

C is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

**2) Machine Independent or Portable:**

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

**3) Mid-level programming language:**

C is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.

**4) Structured programming language:**

C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

**5) Rich Library:**

C provides a lot of inbuilt functions that makes the development fast.

**6) Memory Management:**

It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free () function.

**7) Speed:**

The compilation and execution time of C language is fast.

**8) Pointer:**

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

**9) Recursion:**

In c, we can call the function within the function. It provides code reusability for every function.

**10) Extensible:**

C language is extensible because it can easily adopt new features.

### 3. Explain About Generation of Programming Languages?

#### **Programming Language:**

A programming language is a set of words, symbols and codes that enables humans to communicate with computers.

#### **Generation of Programming Language:**

Generation	Language/type
1	Machine Language
2	Assembly Language
3	Imperative Language
4	Object Oriented Languages
5	Logic Languages

#### **FIRST GENERATION OF PROGRAMMING LANGUAGE:**

The first generation of programming language, or 1GL, is machine language. Machine language is a set of instructions and data that a computer's central processing unit can execute directly. Machine language statements are written in binary code, and each statement corresponds to one machine action.

#### **SECOND GENERATION PROGRAMMING LANGUAGE:**

The second generation programming language, or 2GL, is assembly language. Assembly language is the human-readable notation for the machine language used to control specific computer operations. An assembly language programmer writes instructions using symbolic instruction codes that are meaningful abbreviations or mnemonics. An assembler is a program that translates assembly language into machine language.

#### **THIRD GENERATION PROGRAMMING LANGUAGE:**

The third generation of programming language, 3GL, or procedural language uses a series of English-like words that are closer to human language, to write instructions. High-level programming languages make complex programming simpler and easier to read, write and maintain. Programs written in a high-level programming language must be translated into machine language by a compiler or interpreter. PASCAL, FORTRAN, BASIC, COBOL, C and C++ are examples of third generation programming languages.

#### **FOURTH GENERATION PROGRAMMING LANGUAGE:**

The fourth generation programming language or non-procedural language, often abbreviated as 4GL, enables users to access data in a database. A very high-level programming language is often referred to as goal-oriented programming language because it is usually limited to a very specific application and it might use syntax that is never used in other programming languages. SQL, NOMAD and FOCUS are examples of fourth generation programming languages.

#### **FIFTH GENERATION PROGRAMMING LANGUAGE:**

The fifth generation programming language or visual programming language, is also known as natural language. Provides a visual or graphical interface, called a visual programming environment, for creating source codes. Fifth generation programming allows people to interact with computers without needing any specialised knowledge. People can talk to computers and the voice recognition systems can convert spoken sounds into written words. Prolong and Mercury are the best known fifth-generation languages.

#### 4. Explain about Algorithm/pseudo code.

##### Algorithm:

An algorithm is a method of representing the step – by – step procedure for solving a problem. An algorithm very useful for finding the right answer to a problem or to a difficult problem by breaking the problem into simple cases.

##### Properties of Algorithm:

1. Finiteness
2. Definiteness
3. Effectiveness
4. Generality
5. Input
6. Output

##### 1. Finiteness :

The algorithm must always terminate after a finite number of steps.

##### 2. Definiteness:

Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.

##### 3. Effectiveness:

All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

##### 4. Generality:

The algorithm should be complete in itself so that it can be used to solve all problems of a given type for any input data.

##### 5. Input:

An algorithm has zero or more inputs, taken from a specified set of objects.

##### 6. Output:

An algorithm has one or more outputs, which have a specified relation to the inputs.

##### Advantages of Algorithm:

1. It is a step – by – step solution to given problem, which is very easy to understand.
2. It has got a definite procedure, which can be executed within a set period of time.
3. It is an independent of programming language.
4. It is easy to debug as every step has got its own logical sequences.

##### Example of Algorithm:

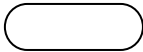




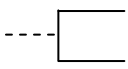

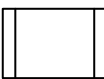
Suppose we want to find the average of three numbers, the algorithm is given as follows:

<b>Step – 1</b>	:	Start
<b>Step – 2</b>	:	Enter the value of a, b and c
<b>Step – 3</b>	:	<b>d</b> =a+b+c
<b>Step – 4</b>	:	Print the value of d
<b>Step – 5</b>	:	Stop

## 5. Explain about Flowchart SYMBOLS.

### Flowchart:

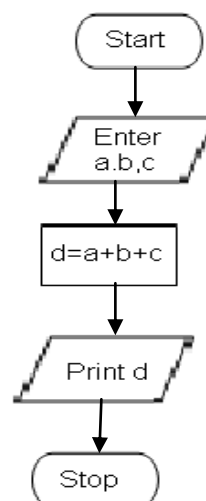
Flowchart is diagrammatic representation of an algorithm. It is built using different types of boxes and symbols. The operation to be performed is written in the box. All symbols are interconnected by arrows to indicate the flow of information and processing.

Oval		Terminal	Start/Stop
Parallelogram		Input/output	Making data available for processing(input) or recording of the processed information (output)
Rectangle		Process	Any processing to be performed.
Diamond		Decision	Decision or switching type of operations that determines which of the alternative path is to be followed.
Circle		Connector	Used for connecting different part of flow chart.
Bracket with broken line		Annotation	Descriptive comments or explanations
Arrow		Flow	Joins two symbols and also represents executions flow.
Double side rectangle		Predefined process	Modules or subroutines given elsewhere

### Advantages of Flow Chart:

1. Conveys Better meaning
2. Analyses the problem effectively
3. Effective Coding
4. Systematic Debugging

### Flow chart Ex:



## 6. Explain about Structure of a C-Programming Language?

"C" is a structured programming language. A "C" program is not more than a collection or set on functions. Each function is a collection of statements. Which performs a specific task in the program.

1	<b>Documentation section</b>	<code>/* Comments about the program*/</code>	
2	<b>Header files section</b>	<code>#include&lt;stdio.h&gt;</code> <code>#include&lt;conio.h&gt;</code> ..... .....	
3	<b>Global declaration section</b>	<code>data_type var1, var2...;</code>	
4	<b>Main program section</b>	<code>main()</code> <code>{</code> <code>data_type var1, var2,.....;</code> <code>Declaration sttements;</code> .....; .....; <code>Executable sttements;</code> <code>}</code>	
5	<b>User-defined function section</b>	<code>user_defined_functions(list of arguments)</code> <code>{</code> <code>Statement-1;</code> <code>Statement-2;</code> .....; .....; <code>Statement-n;</code> <code>}</code>	<b>Not Compulsory</b>

### 1. Documentation section:

The readability of the program, the programmers can provide comments about the program in this section.

**Example:** `/*program printing a line of text*/`

2. **Header files section:** "C" program depends on the header-file to a great extent. A header-file contains the information required by the compiler when calls to the library function used in the program occurs during compilation.

**Example:** `#include<stdio.h>`, `#include<conio.h>`,

3. **Global Declaration Section:** This section used for declaration the global variables.

4. **Main program section :** This section starts with a function named as "main()". Every "C" program must contain only one main() function. It is the starting point for program execution.

### 5. User-defined function section:

"C" language provides the facility for the user to define their own functions. These user - defined functions are defined after the main() function. This section is not mandatory in each "C" Program

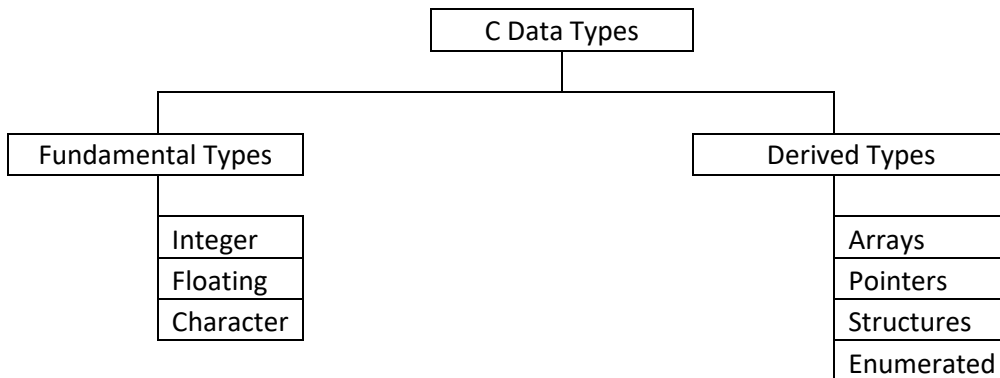
## 7. Explain about data types in "C" Language.

### Data Type:

A data type is used to indicate the type of data value stored in a variable. The data type of a variable is specified at time of its declaration and is attached till the end of execution of program.

In general, data type associated with variable indicates,

- ❖ Type of value stored in the variable.
- ❖ Amount of memory (size) allocated for data value.
- ❖ Type of operations that can be performed on the variable.



- 1) Primary (or) Fundamental Data Types: All C-compilers support four fundamental data types namely Integer (int), Character (char), floating point(float) and Double precision floating point (double).

i) **Integer Type (int):** Integers are whole numbers with a range of values supported by a particular machine. Generally Integers occupy one word of storage and since the word sizes of machines vary the size of an integer that can be stored depends on the size of the integer value is limited to the range – 32,768 to +32,767 (i.e.  $-2^{15}$  to  $2^{15}-1$ ).

C has three classes of integer storage namely *short int*, *int* and *long int* in both signed and unsigned forms. *Short int* represents small integer values and requires half the amount of storage as a regular *int* number uses.

ii) **Floating Point Types (Float):** Floating-point numbers are stored in 32 bits with 6 digits of precision. Floating-point numbers are defined in 'C' by the key word *float*. When the accuracy provided by the float number is not sufficient, the type *double* data type number uses 64 bits giving a precision of 14 digits. These are known as *double* precision numbers. We extend the precision further we may use *long double* which uses 80 bits.

iii) **Character types (char):** A single character can be defined as *character (char)* type data. Characters are usually stored in 8 bits of internal storage. That qualifies signed or unsigned may be explicitly applied to *char*. By unsigned *char* have values between 0 and 255, signed *char* have values from –128 to 127.

## 2) Derived data types:

**I) Array:** A self of adjacent, homogenous memory location that can be stored assign to a name is called an array. An array is a sequence of data in memory, where in all data are of the same type and are placed in physically adjacent locations. Mathematicians think of arrays as matrices with rows and columns. An array is a group of related data items that shares a common name.

Example: `int x[7] = {2,6,9,8,10,3,4}`  
`X[0]=2, x[1]=6, x[2]=9, x[3]=8, x[4]=10, x[5]=3, x[6]=4`

**II) Pointers:** A pointer is known as derived data type. It is derived from built in data types. A pointer variable always contains the address of another variable in it. The address represents memory location of computer memory. A pointer allows us to access and manipulate memory addresses.

**III) Structures:** A structure is a collection data items of different type. Each data item with in structures is said to be a member.

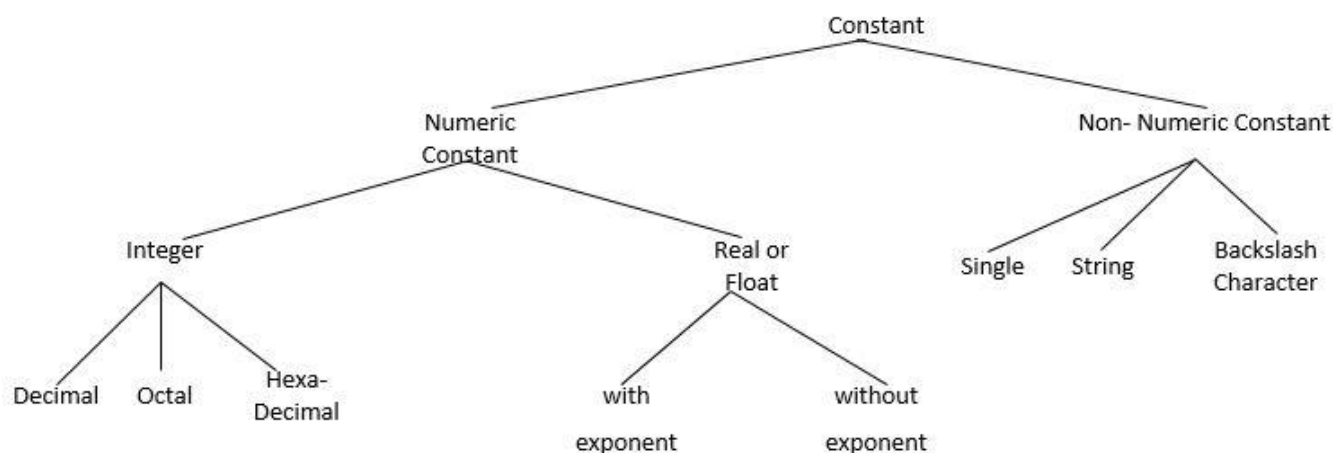
**IV) Enumerated:** another user-defined data type is enumerated data type.

## 8. Explain about Constants In C-Language.

Constants are those quantities whose value does not vary during the execution of the program i.e. value is fixed. Constants in every language are the same.

**In C – Language constants are mainly of two types:**

- I. Numeric Constants                      II. Non-Numeric or Character Constants



### I. Numeric Constant:

These have numeric value having combination of sequence of digits i.e. from 0-9 as alone digit or combination of 0-9 with or without decimal point (precision value having +ve or -ve).

There are further sub-divided into two categories as:

1. Integer Numeric Constant
2. Real or Float Numeric Constant

#### 1. Integer Numeric Constant:

Integer numeric constants have integer data combination of 0-9 without any decimal point or without decimal point precision value having +ve or - ve.



There are further sub-divided into three parts:

**a) Decimal Integer Numeric Constant:**

These have no decimal point in it and is either be alone or be the combination of **0-9** digits. These have either + ve or – ve sign.

For example: 124, -321, 0, 24 etc.

**b) Octal Integer Numeric Constant:**

These consists of combination of digits from **0-7** with + ve or – ve sign. It has leading with O or o (upper or lower case) mean Octal or octal.

For example: O37, O0, -O450 etc.

**c) Hexadecimal Integer Numeric Constant:**

These have hexadecimal data. It has leading ox, OX, Ox, or X, x. These have combination of **0-9 and A – F (a-f)** or alone. These letters a-f or A-F represents the numbers **10-15**.

For example: Ox32, oX92, OxABCD, oXEF etc.

**2. Real or Float Numeric Constant:**

Some constants which have a decimal point or precision value within it having any + ve or – ve sign is called Real Numeric Constant. It is also called Floating Point Constant or float value.

For example: 22.33, -9.8 etc.

**II. Non-Numeric or Character Constants:**

Character constant have either a single character or group of character or a character with backslash used for special purpose.

These are subdivided into three types:

1. Single Character Constant
2. String Character Constant
3. Backslash Character Constant

**1. Single Character Constant:**

These have a single character within single quote ('). So these are also called single quote character constant.

For example: 'a', 'M', '-5', '+ ' etc.

**2. String Character Constant:**

A string is the combination of character or group of characters. A string character constant or a string is enclosed double quotes ("). So it is called 'Double quote' character constant.

For example: "Raja", "Hello", "6622", "5+3" etc.

**3. Backslash Character Constant**

These are used for special purpose in the C – Language. These are used in output statements like **Printf, Scanf** etc. another name Backslash Character Constant is "Escape sequence".

A list of Backslash Character Constant below:

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab

### 9. What is Variables? What are the rules to form a Variable name? How to define them?

A variable is a name that may be used to store a data value. Unlike constant, variables are changeable; we can change value of a variable during execution of a program. A programmer can choose a meaningful variable name.

Example: a, b, average, height, age, total etc.

#### Rules to form a Variable name:

1. Variable name must be up to 8 characters.
2. Variable name must not start with a digit.
3. Variable name can consist of alphabets, digits and special symbols like underscore.
4. Blank or spaces are not allowed in variable name.
5. Keywords are not allowed as variable name.

### 10. What is Identifiers?

In C language identifiers are the names given to variables, constants, functions and user-defined data. These identifiers are defined against a set of rules.

#### Rules for an Identifier:

1. An Identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore (\_).
2. The first character of an identifier can only contain alphabet (a-z, A-Z) or underscore (\_).
3. Identifiers are also case sensitive in C. For example *name* and *Name* are two different identifiers in C.
4. Keywords are not allowed to be used as Identifiers.
5. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

### 11. What is Character set?

Character set is the combination of alphabets or character, digit, special characters and white spaces.

1. Letters (all alphabets a to z & A to Z).
2. Digits (all digits 0 to 9).
3. Special characters, (such as colon, semicolon, period, underscore, ampersand & etc.).
4. White spaces. It has blank space, new line, horizontal tab, vertical tab etc....

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

## 12. What are the Key words in c?

A keyword is a reserved word. You cannot use it as a variable name, constant name etc. There are only 32 reserved words (keywords) in C language?

A list of 32 keywords in c language is given below:

Auto	break	Case	Char	const	continue	default	do
double	else	Enum	extern	float	for	goto	if
Int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

## 13. Write and Explain about "input and output" functions in C-Language.

Input and Output operations are performed using predefined library functions. These are classified into two types:

- i. Formatted I/O functions,
- ii. Unformatted i/o functions

### I. Formatted I/O Functions:

These are two formatted functions.

- a) **Scanf():** Used for formatted input  
This function is used to read values for variables from keyboard.

**Syntax:**

```
Scanf ("control string", address_list);
```

**Example:**

```
Scanf ("%d",&a);
```

- b) **Printf():** Used to obtain formatted output  
This function is used to print result on monitor.

**Syntax:**

```
Printf ("control string", arg1, arg2,..., argn);
```

**Example:**

```
Printf ("%d %c",num,ch);
```

### II. Unformatted i/o functions:

getchar()    putchar()    puts() gets()    getch()    gets()

**getchar():**

This function returns single character entered from keyboard. No arguments are required for this function. By calling this function we can read a string.

**Syntax:**

**Var = getch()**    here, var is an identifier of char type.

**Putchar():**

This function displays a single character on an output device.

**Syntax:**

**Putchar(var);**

**gets ():**

this function reads an input string.

**Syntax:**

**gets (var);**

**puts():**

this function displays a string from keyboard.

**Syntax:**

Puts (var);

**/\*write a c- program using with all input and output statements\*/**

```
# include<stdio.h>
Void main ()
{
    int i=0;
    Char ch[10];
    Char a[25];
    Printf("Enter String:");
    gets(a);
    puts(a);
    do
    {
        Printf ("Enter character from keyboard :");
        i++;
        getchar(ch[i]);
    } while(ch[i]!=EOF);
    For(i=1;ch[i]!='\0';i++)
        Puchar(ch[i]);
}
```

**Output:**

Enter character from keyboard:  
abcd^z

Actual Characters are: abcd  
Enter string: Hello  
Hello

**14. Explain about Operators in C- Language?**

C-Language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. Operators are used in program to manipulate data and variables.

**C-Operators can be classified into following types:**

- ❖ Arithmetic Operators
- ❖ Relation Operators
- ❖ Logical Operators
- ❖ Bitwise Operators
- ❖ Assignment Operators
- ❖ Increment & Decrement Operators
- ❖ Conditional Operators
- ❖ Special Operators

**❖Arithmetic Operators:**

Arithmetic Operators are used for arithmetic operations like Addition, Subtraction, Multiplication, Division etc.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by de-numerator
%	remainder of division

❖ **Relation Operators:**

These create relationship between two operands. Relational operators are used for comparison purpose. The expressions having two operands and one relational operator is called Relational Expression.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

❖ **Logical Operators:**

Logical operators are used for logical operations. These operations are used for compound relational expressions or logical expressions. When more than one relational expressions occur in a c- expression using logical operators, then these are called Compound Relational Expressions. These are used in decision making statement and some looping statements like if, switch, do while, while and for etc....

Operator	Description	Example
&&	Logical AND	(a && b) is false
	Logical OR	(a    b) is true
!	Logical NOT	(!a) is false

❖ **Bitwise operators:**

Bitwise Operators are similar to that of logical operators except that they work on binary bits. When bitwise operators are used with variables, they are internally converted to binary numbers and then bitwise operators are applied on individual bits.

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

### ❖ Assignment operators:

The assignment operator is used to assign the contents of the one variable to another variable. Or a value is assigned to a variable equal symbol ("=") is used as assignment operator in C-Language.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	multiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

### ❖ Increment & Decrement operators:

The increment operator (++) increments the operand by 1, same as the decrement operator (- -) decrement the operand by 1.

Operator	Description
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

### ❖ Conditional operator:

It is also known as ternary operator and used to evaluate conditional expression.

The syntax of it is: **EXP1? EXP 2: EXP3;**

**For example:**

```
if (number % 2 == 0)
    even = TRUE;
else
    even = FALSE;
```

**The can be written as:**

```
even = (number % 2 == 0)? TRUE: FALSE;
```

### ❖ **Special operators:**

There some special operators available in "C", such as comma operator, size of operator, and member selection operator.

<b>Operator</b>	<b>Description</b>	<b>Example</b>
sizeof	Returns the size of an variable	<b>sizeof(x)</b> return size of the variable <b>x</b>
&	Returns the address of an variable	<b>&amp;x ;</b> return address of the variable <b>x</b>
*	Pointer to a variable	<b>*x ;</b> will be pointer to a variable <b>x</b>

## **15. Write and explain about Comment lines in C- Language?**

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in C language.

1. Single Line Comments
2. Multi Line Comments

### **1. Single Line Comments:**

Single line comments are represented by double slash \\.

#### **Syntax:**

```
// code to be commented
```

### **2. Multi Line Comments:**

Multi line comments are represented by slash asterisk \\* ... \*\ . It can occupy many lines of code but it can't be nested.

#### **Syntax:**

```
/* code to be commented */
```

## **16. Write and explain about Escape Sequence in C- Language?**

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \ .

**For example:** \n represents new line.

### List of Escape Sequences in C:

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab

\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

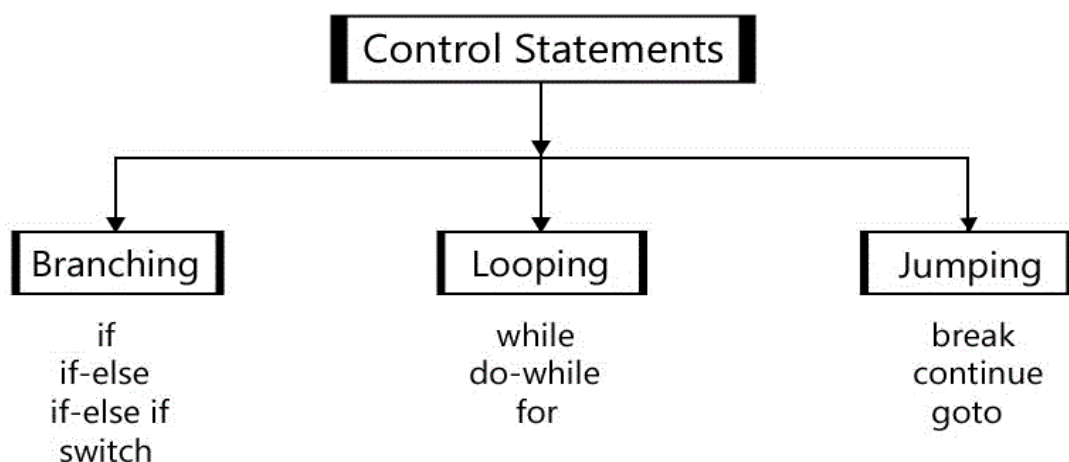
## 17. Write and explain about Flow of Control in C- Language?

### Flow of Control:

In the term software or computer programming, it has a set of instruction called **Program**. These instructions are called statements, which occurs sequentially or either conditional way or in the iterative way. To handle such type of statements, some flow controls required. These flow controls are called **Control Statements**.

These are mainly three types of controls statements or flow controls:

1. **Branching**
2. **Looping**
3. **Jumping**





## 18. Explain about control structures (or) statement using C- program?

The **if statement** in C language is used to perform operation on the basis of condition. By using if-else statement, you can perform operation either condition is true or false.

There are many ways to use if statement in C language:

- |                        |                          |
|------------------------|--------------------------|
| 1) Simple If statement | 3) Nested if statement   |
| 2) If-else statement   | 4) Switch case statement |

### 1. Simple If Statement:

The single if statement in C language is used to execute the code if condition is true. Otherwise it skips that action.

The syntax of simple if statement is given below:

```
if(condition)
{
True - Statement block;
} statement_next;
```

Here the following operations are performed.

- The true statement `_block` may contain single statement or multiple segments. Initially the value of condition is evaluated. If the value is true, `statement-block` is executed followed by `statement_next`.
- If the value of condition is false, `statement_next` is executed directly (`statement-block`) is skipped.

### 2. If-else Statement:

This statement performs an action if condition is true otherwise performs another action. It is used for two way decision – making. This statement also has a single condition, but it has two blocks. One is true block and other is false block...

The syntax of if- else statement is given below:

```
if (Condition)
{
Statement _ block1; //code to be executed if condition is true
}
else
{
Statement _ block2; //code to be executed if condition is false
}
Statement _ next;
```

Here the condition is evaluated, if it is true `statement_block1` is executed followed by `statement_next`, otherwise, (if it is false), `statement_block2` is executed followed by `statement_next`.

### 3. Nested if statement:

There can be any number of nested if statements within an if statement. The if else-if statement is used to execute one code from multiple conditions. Then it is called nested if statement.

The syntax of Nested if statement is given below:

```
if (condition1)
{
    Statement_1; //code to be executed if condition1 is true
}
else if(condition2)
{
    Statement_2; //code to be executed if condition2 is true
}
else if(condition3)
{
    Statement_3; //code to be executed if condition3 is true
}
else
{
    Statement_4; //code to be executed if all the conditions are false
} Statement_ x;
```

Here first of all condition 1 will be checked. If condition 1 is true, then further condition 2 will be checked. If condition 2 is true, then first statement block st1 will be executed and after the execution of st1, statement- x (if exists) will be executed. Similarly if condition becomes false, then condition 3 will be checked.

### 4. Switch Case Statement:

This is a multi-way decision maker that test whether an expression matches one of the number of values. When number of conditions occurs in a given problem so that the use of ladder if statement becomes difficult, then there is a requirement of such type of statement which should have different alternatives of different cases. For this purpose switch statement is used. It is also called Case statement.

Syntax is given below:

```
Switch (expression)
{
case value1:
Statement _ block 1; //code to be executed;
break; //optional
case value2:
Statement _ block 2;//code to be executed;
break; //optional
.....
default:
code to be executed if all cases are not matched;
} Statement _ next;
```

## 19. Explain about Looping Statements in C-Language?

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages

1. **While Loop**
2. **do while Loop**
3. **For Loop**

Suppose that you have to print table of 2, then you need to write 10 lines of code. By using the loop statement, you can do it by 2 or 3 lines of code only.

### Advantage of loops in C:

- 1) It **saves code**.
- 2) It helps to traverse the elements of array (which is covered in next pages).

S.N.O	Loop Type & Description
1	<b><u>while loop</u></b>
	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	<b><u>for loop</u></b>
	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	<b><u>do...while loop</u></b>
	It is more like a while statement, except that it tests the condition at the end of the loop body.

### 1 while loop statement.

While statement or while loop is an entry control loop. In this, first of all condition is checked and if it is true then group of statement or body of loop will be executed. It will execute again and again till condition becomes false.

- In while loop, condition is given before the statement. So it is different from the do while loop. It can execute the statements 0 or more times.
- It is better if number of iteration is not known by the user.

### The syntax of while loop is given below:

```

While (test condition)
{
    Block of statements; //code to be executed
}
Statement - x;
```

**Example program using with while loop in C language:**

/ The simple program of while loop that prints table of 1.\* /

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=1;
clrscr();
while(i<=10)
{
printf("%d \n",i);
i++;
}
getch();
}
```

**Output:**

```
1
2
3
4
5
6
7
8
9
10
```

**2. do-while loop statement.**

do statement is exit control loop. It is also called do – while statement. In this statement, first body of the loop is executed and then the condition is checked. If condition is true, then the body of the loop will be executed. When condition becomes false, then it will exit from the loop.

- In do while loop, statement is given before the condition, so statement or code will be executed at least one time. In other words, we can say it is executed 1 or more times.
- It is better if you have to execute the code at least once.

**The syntax of while loop is given below:**

**do**

{

Block of statements; //code to be executed

} **while** (condition);

Statement – x;

Note that semicolon (;) must be at end of the while condition.

**Example program using with do - while loop in C language:**

There is given the simple program of c language do while loop where we are printing the table of 1.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=1;
clrscr();
do
{
printf("%d \n",i);
i++;
}while(i<=10);
getch();
}
```

**Output:**

```
1
2
3
4
5
6
7
8
9
10
```

### 3. For loop statement.

It is a looping statement, which repeat again and again upto a defined condition. It is one step loop, which initialize, check the condition and increment / decrement step in the loop in a single statement.

#### The syntax of for loop is given below:

```
for(initialization value ;test condition ; increment / decrement)
{
body of the loop;//code to be executed
}
Statement - x;
```

#### **Flowch** Example program using with for loop

##### in C language:

/\*The simple program of for loop that prints table of 1.\*/

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=0;
clrscr();
for(i=1;i<=10;i++)
{
printf("%d \n",i);
}
getch();
}
```

#### **Output:**

```
1
2
3
4
5
6
7
8
9
10
```

### 20. Write and explain about Jumping Statement?

Jumping Statement is also called go to statement. It is sometimes also called part of branching statement. The go to move the controls on a specified address called label or label name.

The go to be mainly of two types. One is conditional and other UN conditional.

Also jump can be either in forward direction or in backward direction.

There are three different controls used to jump from one c – program statement another and make the execution of the programming procedure fast.

There three jumping controls are:

#### **1. break statement 2) Continue statement 3) go to statement**

**1. break statement:** The break statement is used to terminate the control from the loop statements of the switch-case structure. The break statement is normally used in the switch case loop and in each case condition, the break statement must be used it not the control will be transferred to the subsequent case condition also

The general form of the break statement is:

The semicolon must be inserted after the break statement.

### Example:

```

/*The simple program of break
statement*/
#include <stdio.h>
main()
{
    int i;
    clrscr();
    for(i=1;i<=10;i++)
    {
        Printf("The skbr college");
        If(i==5)
        Break;
        printf("i=%d \n",i);
    }
    getch();
}

```

### 2. Continue Statement:

Continue allows is to continue the execution of for or while loop from the beginning, making it possible to skip the rest of the statements in the loop after the continue statement.

```

/*The simple program of continue statement*/
#include <stdio.h>
main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        Printf("The skbr college");
        If(i==5)
        contine;
        printf("Amalapuram");
    }
    getch();
}

```

### 3. goto Statement:

It is used to alter the normal sequence of program execution by transferring the control to some other part of program.

It is written as go to label,

- Label is an identifier which is used to give the location of target statement to which control must be transferred.
- Target statement is given as label: statement;
- Each labelled statement written within a program must have unique label.

## 21. What is function? Explain about function in c Language?

A function definition is the complete description of a function. Actually a function definition tells what a function does and how it performs. A function definition contains a function body (block of statements) in addition to function name, arguments list and return type.

### **General syntax of function definition:**

```
Return- type function – name (parameter - list)
{
    Local variable declarations
    -----
    Statements
    -----
    Return (expression);
}
```

### **Function:**

A function is self-contained program segment that performs specific and well defined tasks. Use of functions cases the complex tasks by dividing the program. Subprogram is the independent and complete program. It is also called function.

### **There are two types function in C-language:**

#### **1. Programmer – defined Functions (or) User Define Functions:**

These functions allows the programmer to define their own functions according to the requirement of the program.

#### **2. Library functions ( or) Built-in Functions:**

These are predefined functions that are provided to perform computations such as main(), sin (x), POW (x, i), sqrt (x) etc... these functions cannot be modified by the programmer.

### **The structure of a function is,**

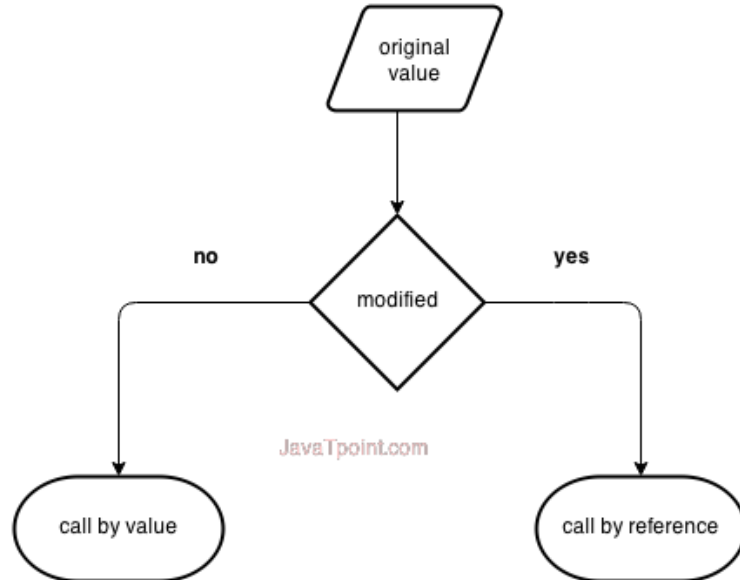
```
Return _ value _type function _name (parameter _ list)
{
    Declarations
    -----
    Statements
    -----
    Return (expression);
}
```

## 22. Explain about Passing parameters (or) Parameter Passing? (pass/call by values and pass/call by reference?)

### Parameter Passing Techniques:

There are two ways of passing parameters to a function.

- (i). Call - by - value
- (ii). Call - by - reference.



### (i). Call - by - value:

When an argument is passed by value, the data in the actual parameters is copied to the formal parameters. Any modification done to formal parameters will not be reflected back to actual parameters.

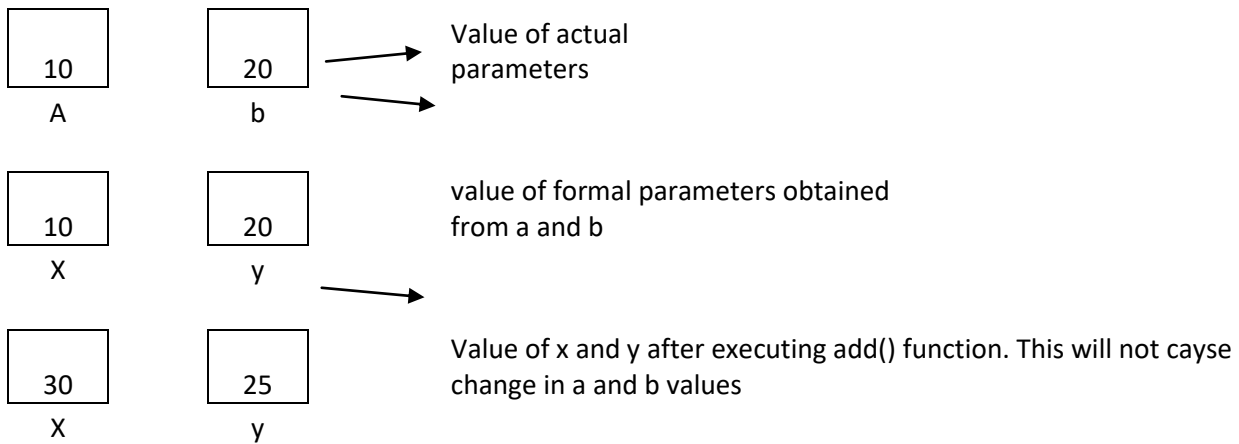
#### ***/\*Program for Call - by - value\*/***

```

#include <stdio.h>
#include <conio.h>
Void add(int, int);
Void main()
{
Int a=10, b=20;
Add(a,d);
Printf("a=%d\t b=%d",a,b);
}
Void add(int x, int y)
{
    X=x+y;
    Y=y+5;
    Printf("x=%d\t y=%d",x,y);
}
  
```



In this program, the values of actual parameters i.e., a and b are passed to function add (int x, int y)



### (ii). Call – by – reference:

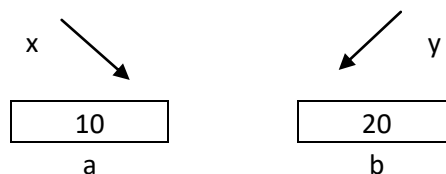
This is also known as call by address. In this mechanism, the address of actual parameters are passed to formal parameters. So, any changes made to the formal parameters causes change in actual parameters also.

```

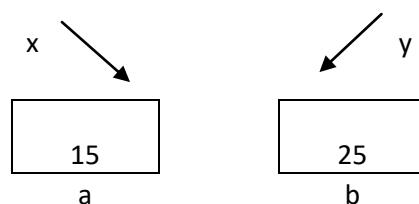
/*Program for Call – by – reference*/
#include <stdio.h>
#include <conio.h>
Void add(int, int);
Void main()
{
  Int a=10, b=20;
  Add(&a, &d);
  Printf("x=%d\t y=%d",a,b);
}
Void add(int *x, int *y)
{
  *X=*x+5;
  *Y=*y+5;
}

```

In this program, the addresses of a and b are passed to x and y. hence they refer to some address location.



After execution of add () function the values at x and y are changed. Hence this also refers to the change of values of a and b.



Value of after add () function is executed.

### Difference between Call by Value and Call by Reference in C:

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

### 23. Explain about different types of functions.

#### Types of functions:

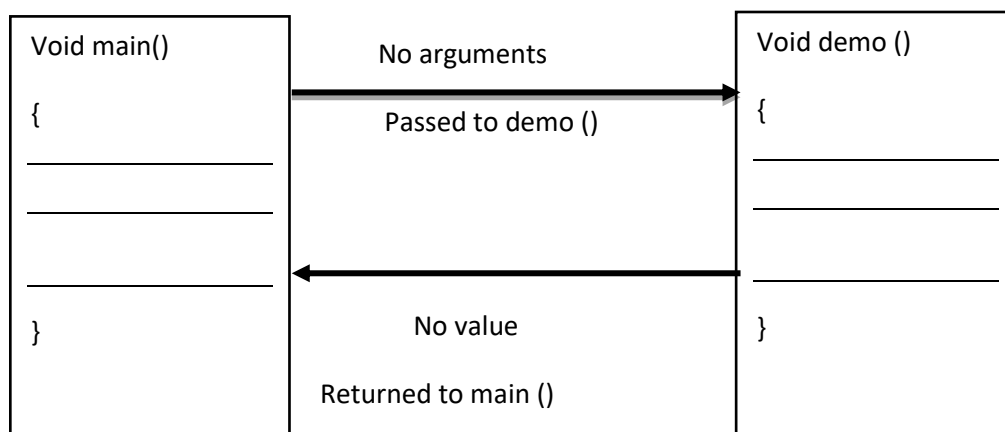
Depending on the return value and argument passed, functions can be classified into four types.

- i. **Function with No Argument and No Return Values**
  - ii. **Function with Argument and No Return Values**
  - iii. **Function with Return Values and No Argument**
  - iv. **Function with Argument and Return Values**
- i. **Function with No Argument and No Return:**

These are the functions in which no parameters are passed from calling function to called function and no values are returned from called function to calling function.

#### **Example:**

```
Void demo ()      /* no arguments*/
{
  Printf ("this is demo program");
}
Void main ()      /*no return statement*/
{
  demo ();
}
```

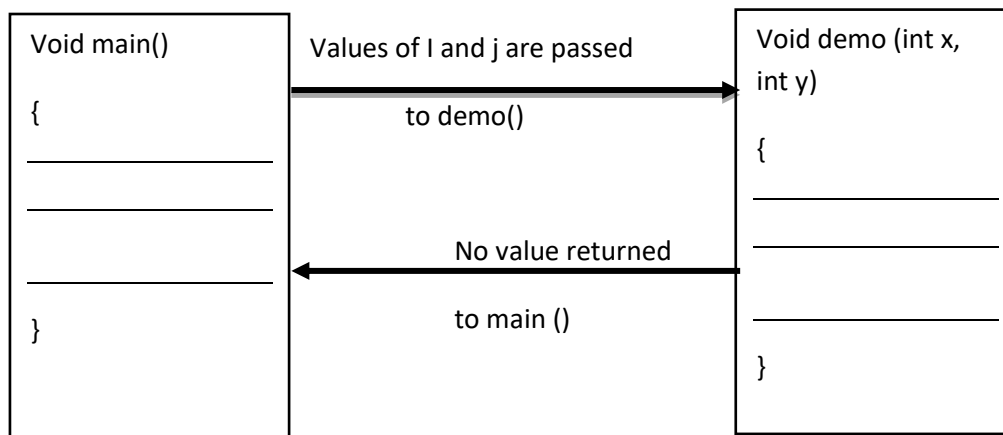


**ii. Function with Argument and No Return Values:**

These are the functions in which arguments are passed from calling function to called function but no values are returned from called function.

**Example:**

```
Void demo (int x, int y) /*it has arguments*/
{
Int s;
S=x+y;
Printf("this is demo program\n sum = %d",s);
} /*no return value*/
Void main()
{
Int i=10,j=5;
Demo(I,j);
}
```

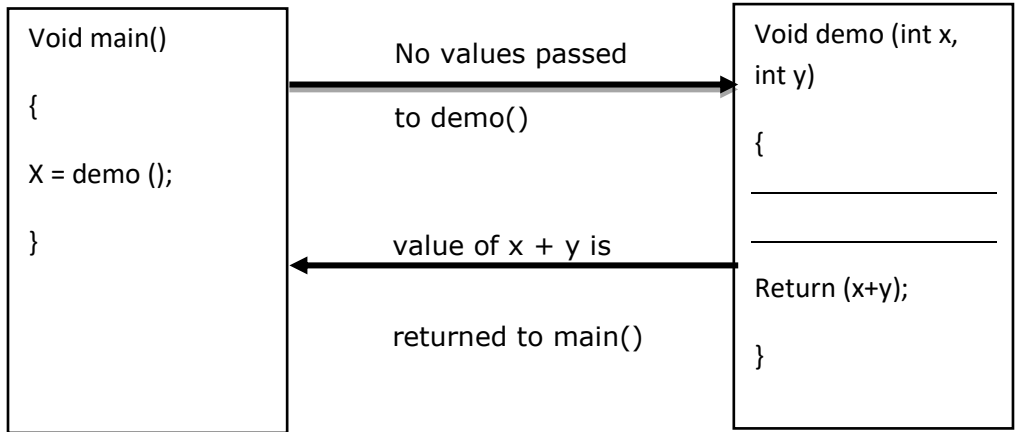


**iii. Function with return values and No Arguments:**

in this no arguments are passed from calling function to called function but values are returned from called function to calling function.

**Example:**

```
Void main()
{
Int x;
Int demo();
X= demo();
Printf ("sum is : %d",x);
}
Int demo(0 /*no arguments*/
{
Int x=10,y = 15;
Printf("this a demo program");
Return(x+y);/*return values of x+y*/
}
```

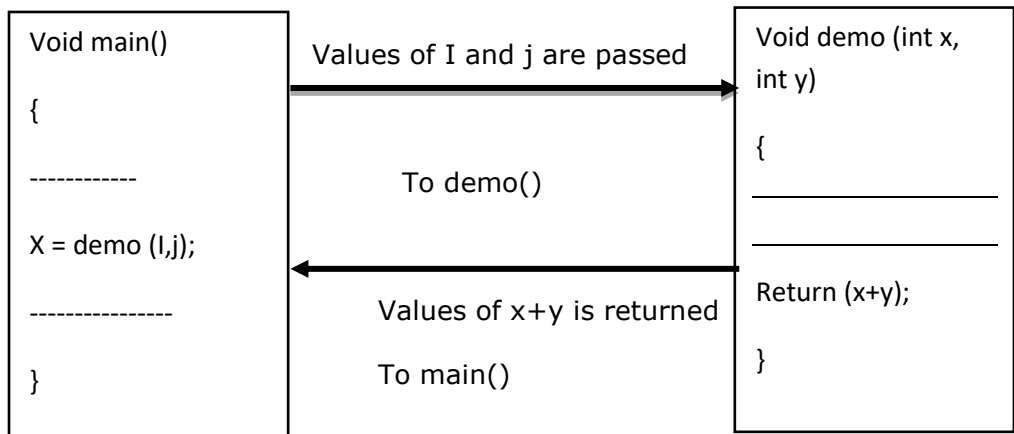


**iv. Function with Argument and Return Vales:**

These are the functions in which parameters are passed form calling function to called function and values are returned from called function to calling function.

**Example:**

```
Void main ()
{
Int i=10,j=5,x;
Int demo (int x, int y);
X= demo (I,j);
Printf("sum is:%d",x);
}
Int demo(int x, int y)
{
Printf("this is a demo program:");
Return(x+y);
}
```



## 24. Write about Recursion Function and types of Recursion function?

### **Recursion: (or) Recursive:**

Recursion is the process or technique by which a function calls itself. A recursive function contains a statement within its body, which calls the same function. Thus, it is also called as circular definition.

(Or)

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

### **Syntax of Recursion:**

```
Void recursion ()
{
    recursion(); /* function calls itself */
}
```

```
Int main ()
{
    recursion();
}
```

### **Types of Recursion functions:**

The word recursive refers to reoccurrence or repeated happening of a function. Hence, recursive functions are those functions which repeatedly calls itself till a prescribed target is achieved.

**There are two kinds namely:**

1. Direct recursion
2. Indirect recursion

### **Direct Recursion:**

In direct recursion, initially a condition is a set. The function calls itself till this condition is satisfied and terminates, once the condition fails.

### **Indirect Recursion:**

Assume that there are two functions "f1" and "f2". Function- f1 is a "called function" whereas Function-f2 is a "calling function". Then, indirect recursive functions are those functions in which function 'f2' calls function 'f1' in turn function 'f1' calls function 'f2'.

## 25. What is Array? Explain the various types of Array.

Array is a collection of variables of same data type. Array is used to store more than one values of same data type under common name. All the variables in an array share one common name.

Number of values that can be stored in an array is fixed.

Array is also known as sequential list because in array all elements are arranged sequentially in memory.

Using array we can represent multiple values in the tabular form means in the form of row and column.

### **Declaration of Array:**

We can declare an array in the c language in the following way.

```
Data_type array_name[array_size];
```

Now, let us see the example to declare array.

```
int marks[5];
```

Here, int is the data\_type, marks is the array\_name and 5 is the array\_size.

### **Initialization of Array:**

A simple way to initialize array is by index. Notice that array index starts from 0 and ends with [SIZE - 1].

1. Marks [0] =80; //initialization of array
2. Marks [1] =60;
3. Marks [2] =70;
4. Marks [3] =85;
5. Marks [4] =75;

80	60	70	85	75
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

### **Initialization of Array**

### **Advantage of Array:**

- 1) Code Optimization: Less code to the access the data.
- 2) Easy to traverse data: By using the for loop, we can retrieve the elements of an array easily.
- 3) Easy to sort data: To sort the elements of array, we need a few lines of code only.
- 4) Random Access: We can access any element randomly using the array.

**Disadvantage of Array:**

1) Fixed Size: Whatever size, we define at the time of declaration of array, we can't exceed the limit. So, it doesn't grow the size dynamically like Linked List which we will learn later.

**Types of Array:**

There are three types of array

- (1) One Dimensional Array
- (2) Two Dimensional Array
- (3) Multi-Dimensional Array

**One Dimensional Array:**

It is simple form of array in which list of elements are stored in contiguous memory locations and accessed using only one subscript. One Dimensional Array can be represented using only one subscript.

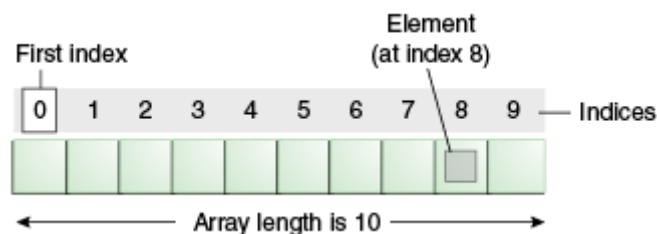
**Syntax:**DataType Array Name [Size];

**Here,**Size indicates number of values that can be stored in an array.

**Example:** int a [10];

Here, 'a' is an array that can store 10 values of type integer.

Individual element of an array can be identify using an array name along with index. Index in an array always starts with 0. So First element is a [0], second is a [1] and so on.

**Initialization one - dimensional Array:**

We can initialize array elements at the place of their declaration itself. The general format of this initialization is:

Type array - name [size] = {list of elements separated by commas}.

**Example:**

```
int a[4] = {5,10,1,55}
```

Here, the size of this array is 4.

5	10	1	55
a[0]	a[1]	a[2]	a[3]

Suppose the list of element specified are less than the size of array then only that many elements are initialized. The remaining elements are garbage values.

### **One Dimensional Array Programs:**

/\*Program to create an array of 10 integer. Enter values for 10 elements and display it.\*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10],i;
    clrscr();
    printf("Enter Elements of Array:\n");
    for(i=0;i<10;i++)
    {
        printf("Enter Value of a[%d]:",i);
        scanf("%d",&a[i]);
    }
    printf("Elements of Array Are:\n");
    for(i=0;i<10;i++)
    {
        printf("a[%d]:%d\n",i,a[i]);
    }
    getch();
}
```

#### **Output:**

Enter Elements of Array:

Enter Value of a[0]:1

Enter Value of a[1]:2

Enter Value of a[2]:3

Enter Value of a[3]:4

Enter Value of a[4]:5

Enter Value of a[5]:6

Enter Value of a[6]:7

Enter Value of a[7]:8

Enter Value of a[8]:9

Enter Value of a[9]:10

Elements of array are:

a[0]:1 a[1]:2 a[2]:3 a[3]:4 a[4]:5

a[5]:6 a[6]:7 a[7]:8 a[8]:9 a[9]:10

### **Two – dimensional array:**

These arrays are also called double dimensional array or two subscript array. These array are in row and column form. So it is also called Row – Column or square Array. These arrays have two subscripts. When data must be stored in the form a matrix we use two – dimensional array.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$$

#### **Syntax:**

Data Type Array Name [Row-Size] [Col-Size];

Here,

Row-Size indicates number of rows in an array.

Col-Size indicates number of columns in an array.

#### **Example:**

```
int a [3] [3];
```

Here, a is an array having three rows and three columns that can store 9 values of type integer.

Also you can declare two dimensional array in static form i.e these type of array declaration have ⇒ fixed or constant values during declaration.



**The syntax used is as follows:**

Static data – type array – name [row size] [column size] = {list value};

**Syntax:**

DataType Array Name [Row-Size] [Col-Size] = {Values};

**Example:**

```
int a[3][3]={1,2,3,4,5,6,7,8,9};
```

**Two Dimensional Array Programs:*****/\*Program to create an array of 2 X 2. Enter values and display it.\*/***

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[2][2],i,j;
    clrscr();
    printf("Enter Elements of Array:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter Value of a[%d][%d]:",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("Elements of Array Are:\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

**Output:**

Enter Elements of Array:

```
Enter Value of a [0] [0]:10
Enter Value of a [0] [1]:20
Enter Value of a [1] [0]:40
Enter Value of a [1] [1]:50
Enter Value of a [2] [0]:70
Enter Value of a [2] [1]:80
```

Elements of Array Are:

```
10 20
40 50
70 80
```

## Multi – dimensional Array:

This type of array has n size of rows, columns and space and so on.

### The syntax used for declaration of this type of array is as follows:

Data – type array – name [s1] [s2] ... .. [Sn];

In tis S n is the nth size of the array. This type of array is less used in c – Language.

#### **Example:**

```
int a[2][3][4];
```

Here,

'a' is an array that can store 24 values of type integer.

#### **Example:**

```
int a[2][3][4];
```

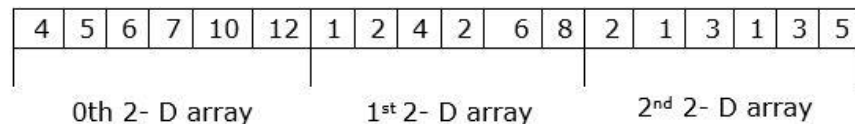
Here,

a is an array that can store 24 values of type integer.

### Initialization of Three – dimensional Array:

```
Int a [3] [2] [3] = {{{[4, 5, 6], [7, 10, 12]}; --> end of 0th 2- D array
                    {[1, 2, 4], [2, 6, 8]}; --> end of 1st 2- D array
                    {[2, 1, 3], [1, 3, 5]}; --> end of 2nd 2- D array}
```

### Physically these elements are stored in memory as shown below:



### **/\*Program to create an array of 3 X 3. Enter values and display it.\*/**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3],i,j;
    clrscr();
    printf("Enter Elements of Array:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter Value of a[%d][%d]:",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("Elements of Array Are:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    getch();
}
```

#### **Output:**

```
Enter Elements of Array:
Enter Value of a[0][0]:10
Enter Value of a[0][1]:20
Enter Value of a[0][2]:30
Enter Value of a[1][0]:40
Enter Value of a[1][1]:50
Enter Value of a[1][2]:60
Enter Value of a[2][0]:70
Enter Value of a[2][1]:80
Enter Value of a[2][2]:90
Elements of Array Are:
1020 30
4050 60
70 80 90
```

## 26. What is String? What are the types of string handling functions?

String is a collection of characters. In terms of C language we can say that a string is an array of characters. String is always enclosed between double quotation mark.

### String Handling Function:

There are lots of string handling functions available in “string.h” header file. Here are some of the important string manipulation functions.

Function	Purpose
<b>strlen()</b>	strlen() function is used to find length of string. Length of string means number of characters in the string. strlen() function accept a string as an argument and returns an integer number which indicates length of string. <b>Syntax:</b> length = strlen (StringName);
<b>strcpy()</b>	strcpy() function is used to copy one string into another string. strcpy() function accepts two string (source and destination) as an argument and copy source string into destination string. It does not return any value. <b>Syntax:</b> strcpy(Destination String, Source String);
<b>strcat()</b>	strcat() function is used to append (join) one string at the end of another string. strcat function accepts two strings (string1 and string2) as an argument and append string2 at the end of string1 and result is stored in string1. It does not return any value. <b>Syntax:</b> strcat (String1, String2);
<b>strcmp()</b>	strcmp() function is used to compare one string with another string. strcmp() function accepts two strings (string1 and string2) as an argument and returns one of the following three integer values: 0 if both string are equal. >0 if string1 is greater then string2. <0 if string1 is less then string2. <b>Syntax:</b> answer = strcmp (String1, String2)
<b>strrev()</b>	strrev() function is used to reverse given string. strrev() function accepts a string as an argument and reverse the string. <b>Syntax:</b> strrev (String);
<b>strstr()</b>	strstr() function is used to search one string into another string. strstr() function accepts two strings (Original String and Search String) as an argument. It returns NULL if search string is not found in Original String. <b>Syntax:</b> strstr (Original String, Search String);

## 27. Explain About Structures and Functions?

The function is a self contained block of instructions.

The heart of effective problem solving is problem decomposition. Taking a problem and breaking it into small managerial pieces is critical to writing large programmes. In 'C' the function construct is used to implement this top down method of programming.

**Function Definition:** The 'C' code that describes what a function does is called the function definition. A function definition has the following general form

```

Type function name (parameter list)
{
  declarations
  statements
}

```

Every thing before the first brace comprises the header of the function definition, and every thing between the braces comprises the body of the function definition. The parameter list is a comma separated list of declarations.

Ex: int factorial (int n)

```

{
  int l, product =1;
  for (i=2; i<=n; i++)
  product = product * i;
}

```

**The Return statement:** The *return* statement may or may not include an expression. The general form is

*return;* (or) *return (variable);* (or) *return (expression);*

When a *return* statement is encountered, execution of the function terminated and control is passed back to the calling statement. If the *return* statement contains a variable or an expression or empty.

```

int factorial (int n)
{
  int i, product = 1;
  for (i =2; i<=n; i++)
  product = product * i ;
  return product;
}

```

**Function prototype:** Function should be declared before they are used 'C' provides for a new function declaration syntax called the *function prototype*. A function prototype tells the compiler the number and type of arguments that are to be passed to the function and the type of the value that is to be returned by the function. The general form of a function prototype is

*type* function name (parameter list);

Ex: double sqrt (double);

This tells the compiler that sqrt () is a function that takes a single argument of type *double* and returns a *double*.

**Call-by-value:** Functions are invoked by writing their name and an appropriate list of arguments within parenthesis. Typically these arguments match in numbers and type the parameters in the parameter list in the function definition. The compiler enforces type compatibility when function prototypes are used. All arguments are passed call-by-value. This means that each argument is evaluated and its value is used locally in place of the corresponding former parameter, thus if a variable is passed to a function that stored value of that variable in the calling environment will not be changed.

## 28. Explain about features of pointer:

A pointer is known as derived data type. It is derived from built in data types. A pointer variable always contains the address of another variable in it. The address represents memory location of computer memory. A pointer allows us to access and manipulate memory addresses.

### Following are the features of pointer:

- (1) Pointer allows Dynamic Memory management facility to the C Programmer.
- (2) Pointer increases the execution speed of the program.
- (3) Pointer Save memory space.
- (4) Pointer allows programmer to directly access and manipulate memory addresses. So it is widely used to represent dynamic data structure such as stack, queue, linked list, tree etc...
- (5) Pointer allows programmer to pass address of the original variable to the function, so function can works with original variable inside function instead of dummy variables (Call by Reference).
- (6) Array and string, can be effectively handled using pointers.
- (7) Pointers are also used for file handling.

### Declaration of Pointer:

#### The general syntax for declaring pointer variable is given below:

Data Type \*Pointer\_Name;

**Here,** Data Type indicates the Data Type of the variable whose address can be stored in Pointer variable.

\* indicates that the variable that is followed is pointer variable.

**Pointer\_Name** is a Pointer Variable that can be used to store address of variable of Type Data Type.

#### **Example:**

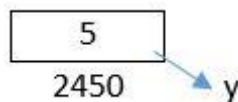
```
int *ptr;
```

**Here,** ptr is a pointer variable that can hold the address of variables of type int.

If you want to store the address of variable of type float then you can declare a pointer variable as shown below:

```
float *ptr;
```

```
int y = 5;
```



### Initialization of Pointer Variable:

Initialize pointer variable means to assign an address of particular variable to pointer variable. Pointer variable can be initialized using **& operator**.

The general syntax for initialize pointer variable is given below:

```
Pointer_Name = & Variable_Name;
```

**Example:**

```
int *ptr;
int x;
ptr = &x;
```

we can also initialize pointer variable at the time of its declaration as shown below:

```
int x;
int *ptr = &x;
```

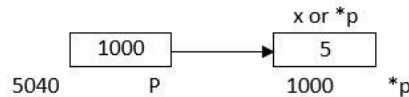
**Access Variable using Pointer:**

Once a pointer variable is declared and initialized, we can access the value of that variable using pointer variable and \* **operator**. \* **Operator is known as indirection or dereferencing operator in C.**

**Consider Following Example:**

```
Int *p;
Int x = 5;
P = &x;
```

Here, address of variable X is assigned to pointer p.

**Pointer and Function Arguments:**

The parameters passed from a calling function can be a 'value' or an address. Whenever a value (i.e., int, float, etc.) is passed, the actual parameters are copied to the formal parameters. The changes made to these values in the called function are not reflected back to the original values (i.e., call - by - value).

Whereas, when an address is passed, the changes made in the called function are reflected back to the original vales (i.e., call - by - reference).

The corresponding formal argument in the called function should be declared as pointer variables.

**29. Explain about arrays of pointers:**

A strong relationship exist between pointers and arrays. Any operation that can be done using array indexing can also be achieved with pointers. Array is a collection of variables of same data type. Thus array is used to store more then values of same data types.

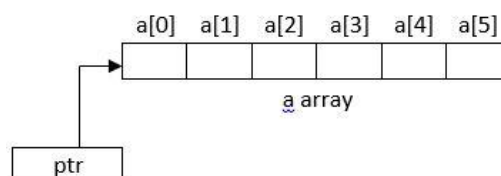
Similarly array of pointer is a collection of addresses. This contains addresses of more then one variables of same data types inside it.

**Consider the declaration pointers using arrays:**

```
Int a [6], *ptr;
Ptr = &a [0];
```

Here, the first declaration defines an integer array of size 6 and pointer to an integer. The second declaration assigns the base address of array 'a' to the 'ptr' variables. The base address of 'a' can also be assigned as ptr = a;

This is shown in figure below.



### 30. Explain about Dynamic Memory Management Functions.

#### Dynamic Memory Management Functions:

In C, the exact size of array is unknown until compile time, i.e., the time when a compiler compiles your code into a computer understandable language. So, sometimes the size of the array can be insufficient or more than required.

Dynamic memory allocation allows your program to obtain more memory space while running, or to release it if it's not required.

In simple terms, Dynamic memory allocation allows you to manually handle memory space for your program.

Although, C language inherently does not have any technique to allocate memory dynamically, there are 4 library functions under "stdlib.h" for dynamic memory allocation.

S.N.	Function & Description
1	<b>void *calloc(int num, int size);</b> This function allocates an array of num elements each of which size in bytes will be size.
2	<b>void free(void *address);</b> This function releases a block of memory block specified by address.
3	<b>void *malloc(int num);</b> This function allocates an array of num bytes and leave them uninitialized.
4	<b>void *realloc(void *address, int newsize);</b> This function re-allocates memory extending it upto newsize.

### 31. Explain about pointers to pointers:

#### Pointers to Pointers:

Pointers to a variable means the pointer contains the address of a variable. Like this a pointer can contain the address of other pointer variable. The pointer variable that contains the address of another pointer is called pointer to pointer.

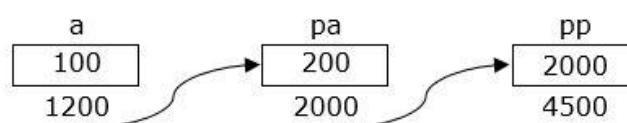
For example, the following declarations defines as integer variable "a", a pointer to an integer "pa" And pointer to a pointer to an integer "pp".

```
Int a;           //integer variable
Int *pa;        //pointer to an variable
Int **pp;       //pointer to a pointer to an integer
```

Suppose the variables 'a', 'pa' and 'pp' are stored as memory locations 1200, 2000, 4500 respectively. And the statements are as follows.

```
Int a, *pa, **pp;
A = 100;
Pa = &a;
Pp = &pa;
```

Then the contents of these variables is shown below:



## Advantages and disadvantages of pointers in c

### Benefits (use) of pointers in c:

1. Pointers provide direct access to memory
2. Pointers provide a way to return more than one value to the functions
3. Reduces the storage space and complexity of the program
4. Reduces the execution time of the program
5. Provides an alternate way to access array elements
6. Pointers can be used to pass information back and forth between the calling function and called function.
7. Pointers allows us to perform dynamic memory allocation and de-allocation.
8. Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
9. Pointers allows us to resize the dynamically allocated memory block.
10. Addresses of objects can be extracted using pointers

### Drawbacks of pointers in c:

1. Uninitialized pointers might cause segmentation fault.
2. Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
3. Pointers are slower than normal variables.
4. If pointers are updated with incorrect values, it might lead to memory corruption.

Basically, pointer bugs are difficult to debug. Its programmer's responsibility to use pointers effectively and correctly.

## 32. Explain about Union in C Language?

A union is a collection of data items of different data types. It's like structures, in which the individual data types may differ from each other.

All the members of the union share the same memory / storage area in the memory.

Every member has unique storage area in structures. In this context, unions are utilized to observe the memory space.

When all the values need not assign at a time to all members, unions are efficient. Unions are declared by using the keyword union, just like structures.

A union is declared as shown below:

```
Union
{
    Datatype member 1;
    Datatype member 2;
    ...      ...      ...
    ...      ...      ...
    Datatype member n;
};
```

```
Example:
union Student
{
    int rollno;
    char s_name[20];
    float average;
};
```

```
/*EXAMPLE PROGRAM FOR C UNION*/
```

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

```
#include <string.h>
```

```
union student
{
    char name[20];
    char subject[20];
    float percentage;
};
```



```

int main()
{
    union student record1;
    union student record2;

    // assigning values to record1 union variable
    strcpy(record1.name, "Raju");
    strcpy(record1.subject, "Maths");
    record1.percentage = 86.50;

    printf("Union record1 values example\n");
    printf(" Name      : %s \n", record1.name);
    printf(" Subject   : %s \n", record1.subject);
    printf(" Percentage : %f \n\n", record1.percentage);
    // assigning values to record2 union variable
    printf("Union record2 values example\n");
    strcpy(record2.name, "Mani");
    printf(" Name      : %s \n", record2.name);

    strcpy(record2.subject, "Physics");
    printf(" Subject   : %s \n", record2.subject);

    record2.percentage = 99.50;
    printf(" Percentage : %f \n", record2.percentage);
    return 0;
}

```

**OUTPUT:**

```

Union record1 values example
Name :Raju
Subject :Maths
Percentage : 86.500000;
Union record2 values example
Name : Mani
Subject : Physics
Percentage : 99.500000

```

33. What is a file? Explain the basic operations performed on a file.

**File:**

A file is collection of records. Each record provides an information to the user. These files are arranged on the disk. These files can be accessed through file handling functions provided by standard "C" library.

**We can perform following operations on file:**

- (1) Create file
- (2) Open file
- (3) Read data from file
- (4) Write data to file
- (5) Close file

**File Management Functions:****1. fopen ()**

fopen () function is used to create a new file or open an existing file. The general syntax for opening a file is given below:

**FILE \*fp;**

**fp= fopen ("filename", "mode");**

Here, FILE is a structure which is defined inside the I/O Library. fp is a pointer of type FILE. Filename is the name of the file which we want to create or open

Mode indicates in which mode you want to open the file.

**Various Modes are:**

r:read  
w:write  
a:append

**2. fclose ()**

fclose () function is used to close the file.

The general syntax for closing the file is given below:

**Fclose (fp);**

Here fp is a pointer using which we have opens the file. It is necessary to close the file to prevent the misuse of the file.

**3. fprintf ()**

This function is used to write set of data to the file.

We can write characters, integer, float etc to the file using this function.

**Syntax:**

fprintf (Filepointer, "control string", list);

Here, Filepointer is the pointer using which we have opens the file. Control string contains output specification for the items in the list. The list may include variable, constants and string

**Example:**

```
char name[20];
int age;
FILE *Fp;
Fp=Fopen ("abc.txt","w");
printf ("Enter Name :");
scanf ("%s",name);
printf("Enter Age:");
scanf("%d",&age);
fprintf (fp,"%s%d",name,age);
fclose (Fp);
```

**4. fscanf ()**

This function is used to read set of data from the file. We can read characters, integer, float etc from the file using this function.

**Syntax:**

fscanf (Filepointer, "control string", list);

Here, File pointer is the pointer using which we have opens the file.

Control string contains input specification for the items in the list. The list may include variable, constants and string

**Example:**

```
char name[20];
int age;
FILE *Fp;
Fp=Fopen ("abc.txt","r");
printf ("Enter Name :");
```

```
scanf ("%s",name);
printf("Enter Age:");
scanf("%d",&age);
fprintf (fp,"%s%d",name,age);
fclose (Fp);
Fp=Fopen ("abc.txt","r");
fscanf(fp,"%s%d",name,&age);
printf("Name=%s\n",name);
printf("Age=%d",age);
fclose(fp);
```

### 5.putc ()

This function is used to write a single character to the file.

#### Syntax:

```
putc (character, Filepointer);
```

Here, Character is the character which we want to write into the file. Filepointer is the pointer using which we opens the file.

#### Example:

```
FILE *Fp;
Fp = fopen ("abc.txt", "w");
putc ('a', Fp);
fclose (Fp);
```

### 6.getc ()

This function is used to read a single character from file.

#### Syntax:

```
Character = getc (File pointer);
```

Here, Character is the character which we want to read from the file. File pointer is the pointer using which we have opens the file.

#### Example:

```
char c;
FILE *Fp;
Fp = fopen ("abc.txt", "r");
c = getc (Fp);
Printf ("Character is %c" , c);
fclose (Fp);
```

### 7.fputc ()

This function is similar to putc () function. This function is used to write a single character to the file.

#### Syntax:

```
fputc (character, Filepointer);
```

Here, Character is the character which we want to write into the file. File pointer is the pointer using which we opens the file.

**Example:**

```
FILE *Fp;
Fp = fopen ("abc.txt", "w");
fputc ('a', Fp);
fclose (Fp);
```

**8.fgetc ()**

This function is similar to `getc ()` function. This function is used to read a single character from file.

**Syntax:**

```
Character = fgetc (Filepointer);
```

Here, Character is the character which we want to read from the file. Filepointer is the pointer using which we have opens the file.

**Example:**

```
char c;
FILE *Fp;
Fp = fopen ("abc.txt", "r");
c = fgetc (Fp);
Printf ("Character is %c" , c);
fclose (Fp);
```

**9.fseek ()**

This function is used to move the cursor (pointer) to the desired position within the file. If the operation performed using `fseek ()` is successful then it returns zero otherwise it returns negative value.

**Syntax:**

```
fseek (Filepointer, offset, Position)
```

Here, File pointer is a pointer using which we opens the file. Offset is a number or variable of type long. The offset specifies the number of position to be moved from the location specified by position. The offset may be positive (move forward) or negative (move backward).Position is an integer number.

**The position may have following values:**

```
0means Beginning of file
1means current position
2means end of file
```

**10.ftell ()**

This function is used to retrieve the current position of the cursor (pointer) in the file. It returns a value of type long indicating position of the pointer within file.

**Syntax:**

```
int n;
n=ftell (Filepointer);
Here, File pointer is a pointer using which we opens the file.
```

## 11.feof ()

This function is used to determine whether the end of the file has been reached or not. When a file is being read sequentially one line, or one piece of data at a time this function can be used to check end of the file has come or not. This function returns nonzero value when the end of file detected and zero value otherwise.

### Syntax:

```
feof (Filepointer);
```

Here, Filepointer is a pointer using which we open the file.

### Example:

```
if (!feof(Fp))
printf("End of File");
```

## 34. Command Line Argument?

The process of passing arguments to the main function while executing the program is known as command line argument.

The general form of main function that accepts command line arguments is given below:

```
main (int argc, char * argv [])
```

Here,

**argc** means **argument counter**. It contains number of arguments passed to the main function while executing the program.

**argv** means **argument vector**. It is an array of character pointers that contains actual arguments that are passed to the main function while executing the program.

The first argument is contained in argv [0], second argument is contained in argv [1] and so on.

The First argument is always the path name of the program that you want to execute.

## 35. Explain about Sequential and random file access in C?

There are two types of files. They are

1. Data file (Eg. record.dat)
2. Text file (Eg. info.txt)

### Data file:

Data file can be used to store information in the form of records. For example, consider a structure with 3 data members (rollno, name and age). We can store structure objects in the form of records in data file as shown below.

```
+-----+
| 101   Tom       10   | Record 1
+-----+
| 102   Raj       10   | Record 2
+-----+
| 103   Taj       11   | Record 3
+-----+
```

**Text file:**

Text file is used to store information in the form of characters as shown below.

```
jp@jp-VirtualBox:~/ $ cat input.txt
one Hello world 12 dsf
one Hello world 123
Hello worldad a
```

Here, input.txt is a text file.

**There are two type of file access.  
They are**

1. Sequential file access
2. Random file access

**Sequential file access:**

Data are stored one after another (sequentially) in the case of text files. So, data are accessed in the same way in which they are stored.

**Random file access:**

Data can be read or modified randomly in data file, since the information are stored in the form of records. By moving file pointer, we can access any needed information from a file. To read last record of a file, we don't need to read through the entire file. Instead, we can move the file pointer to last record and access it. Random file access is more efficient when compared to sequential file access.

**Explain about Different file opening modes in c.****Various file opening modes in C:**

Below are the three basic modes for opening a file.

1. Read mode
2. Write mode
3. Append mode

**1. Read mode:**

This mode opens an existing file for reading. If the given file does not exists, it returns NULL to the file pointer.

**Syntax:**

```
FILE *fp;
fp = fopen("filename.txt", "r");
if (fp == NULL)
    printf("The given file name does not exists\n");
```

where "filename.txt" is the file name and "r" is the mode(read mode).

**2. Write mode:**

This mode opens a new file on the disk for writing. If a file already exists with the given file name, then this mode will overwrite the file.

**Syntax:**

```
fp = fopen("write.txt", "w");
where "write.txt" is the file name and "w" is the mode.
```

### 3. Append mode:

This mode opens a preexisting file for appending data. If the file does not exist, it opens a new file for write operation.

#### Syntax:

```
fp = fopen("append.txt", "a");
```

Where "append.txt" is the file name and "a" is the mode.

Below are few other file opening modes available in C language.

w+	Open for reading and writing. The file is created if it doesn't exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
r+	Open for reading and writing. The stream is positioned at the beginning of the file.
a+	Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

### 36. What is the detecting the End – of – File?

EOF is a condition which is encountered when there is no data left to be read from a file or stream. It is used to mark the end of data, which is usually control Z in reference system. In C, different input / output functions returns a value.

“EOF” macro to indicate end – of – file condition has occurred. The actual values of EOF is system dependent. The reading should be terminated when EOF is encountered.

EOF is useful while transmitting data. Since files are stored in blocks, EOF act as a marker that let the system know if it has allocated enough space to store the file.

### 37. Explain about Remove and Renaming a file in C?

In two previous tutorials we talked about file I/O functions on text files and file I/O functions on binary files. In this C programming language tutorial we look at how to remove files and how to rename a file.

#### Remove a File:

With function remove() we can delete a file. Let us take a look at an example:

```
#include<stdio.h>
int main(void)
{
    char buffer[101];

    printf("Name of file to delete: ");
    gets_s(buffer, 100);

    if(remove(buffer) == 0)
        printf("File %s deleted.\n", buffer);
    else
        fprintf(stderr, "Error deleting the file %s.\n", buffer);
}
```

First a buffer is made with room for 100 characters, plus '\0'. After the question "Name of file to delete:" is printed on the screen the gets\_s is waiting for input of a maximum of 100 characters. (It is also possible to use gets(buffer) but gets\_s is more secure, because you can't overflow the buffer.)

In the "if statement" the function `remove` is used to remove the file. If the file is successfully deleted then a success message is printed. If something goes wrong (for example the file doesn't exist) an error message is printed on the standard error stream.

### **Renaming a File:**

With the function `rename()` we can rename a file. Let us take a look at an example:

```
#include<stdio.h>
int main(void)
{
    char buffer_old[101], buffer_new[101];
    printf("Current filename: ");
    gets_s(buffer_old, 100);
    printf("New filename: ");
    gets_s(buffer_new, 100);
    if(rename(buffer_old, buffer_new) == 0)
    {
        printf("%s has been rename %s.\n", buffer_old, buffer_new);
    }
    else
    {
        fprintf(stderr, "Error renaming %s.\n", buffer_old);
    }
}
```

As in the previous example a buffer for the filename is created of 100 characters, plus a character for '\0'.

This time there is also a second buffer created that will hold the new name.

In the "if statement" the function `rename()` is used. The function `rename()` needs two variables, one for the old name and one for the new file name. If the renaming either succeeds or fails, a message is printed.

## **38. Explain About Error Handling in "C"?**

### **Types of program errors:**

1. Syntax errors: errors due to the fact that the syntax of the language is not respected.
2. Semantic errors: errors due to an improper use of program statements.
3. Logical errors: errors due to the fact that the specification is not respected.

### **From the point of view of when errors are detected:**

1. Compile time errors: syntax errors and static semantic errors indicated by the compiler.
2. Runtime errors: dynamic semantic errors, and logical errors, that cannot be detected by the compiler (debugging).

### **Semantic errors:**

Semantic errors indicate an improper use of Java statements.  
Let us see some examples of semantic errors.

### **Logical errors:**

Logical errors are caused by the fact that the software specification is not respected. The program is compiled and executed without errors, but does not generate the requested result.

Let us see some examples of logical errors:



### Errors detected by the compiler and runtime errors:

All syntax errors and some of the semantic errors (the static semantic errors) are detected by the compiler, which generates a message indicating the type of error and the position in the Java source file where the error occurred (notice that the actual error could have occurred before the position signalled by the compiler).

Other semantic errors (the dynamic semantic errors) and the logical errors cannot be detected by the compiler, and hence they are detected only when the program is executed. Let us see some examples of errors detected at runtime:

Program errors and exception handling 3

### 39. Explain the Storage Classes in C Language

There are four storage class modifiers used in C which determine an identifier's storage duration and scope.

**auto**  
**static**  
**register**  
**extern**

An identifier's storage duration is the period during which that identifier exists in memory. Some identifiers exist for a short time only, some are repeatedly created and destroyed and some exist for the entire duration of the program. An identifier's scope specifies what sections of code it is accessible from.

**Auto:** The auto storage class is implicitly the default storage class used and simply specifies a normal local variable which is visible within its own code block only and which is created and destroyed automatically upon entry and exit respectively from the code block.

**Register:** The register storage class also specifies a normal local variable but it also requests that the compiler store a variable so that it may be accessed as quickly as possible, possibly from a CPU register.

**Static:** The static storage class causes a local variable to become permanent within its own code block i.e. it retains its memory space and hence its value between function calls.

**Extern:** When applied to global variables the static modifier causes them to be visible only within the physical source file that contains them i.e. to have file scope. Whereas the extern modifier which is the implicit default for global variables enables them to be accessed in more than one source file.

For example in the case where there are two C source code files to be compiled together to give one executable and where one specific global variable needs to be used by both the extern class allows the programmer to inform the compiler of the existence of this global variable in both files.