UNIT - I

C: It is a structured programming language that is machine-independent and extensively used to write various applications, Operating Systems like Windows, and many other complex programs like Oracle database, Git, Python interpreter, and more.

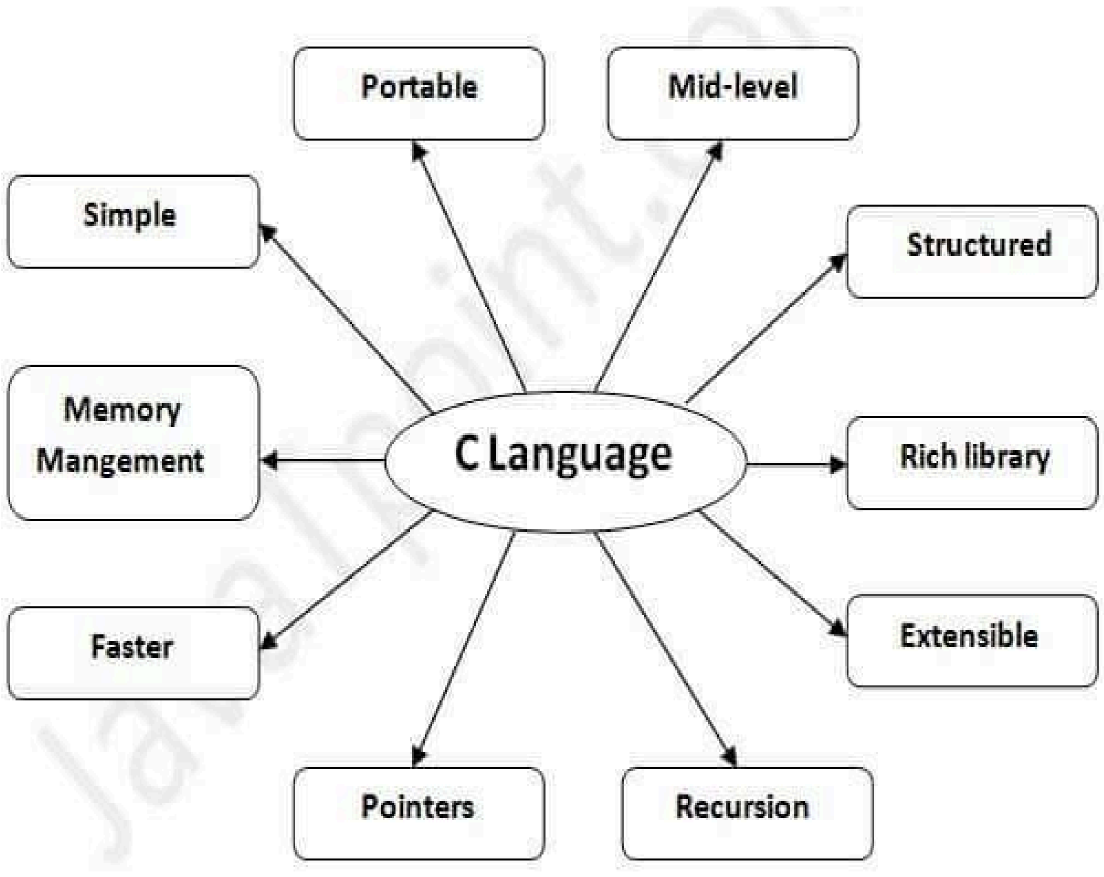
History of C Language

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A. **Dennis Ritchie** is known as the **founder of the c language**. It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

Features of C Language

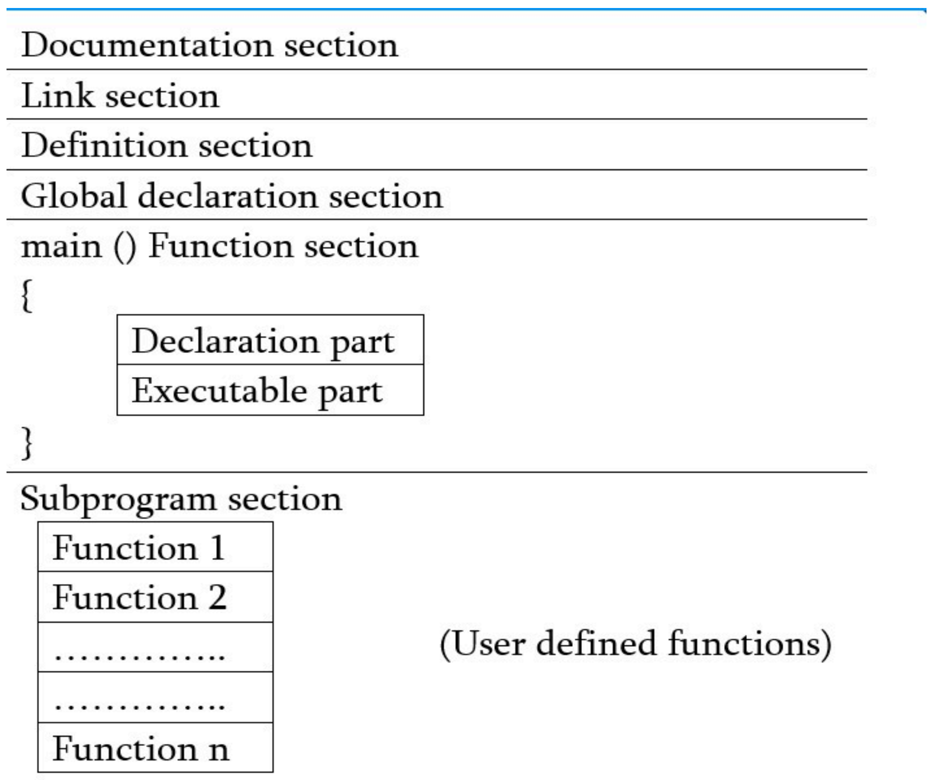


C is the widely used language. It provides many **features** that are given below.

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

Structure of a C

A C program is divided into different sections. There are six main sections to a basic c program.



Documentation Section

The documentation section is the part of the program where the programmer gives the details associated with the program. He usually gives the name of the program, the details of the author and other details like the time of coding and description. It gives anyone reading the code the overview of the code.

Example

```
/**
* File Name: Helloworld.c
* Author: Manthan Naik
* date: 09/08/2019
* description: a program to display hello world
*                   no input needed
```

*/

Link Section

This part of the code is used to declare all the header files that will be used in the program. This leads to the compiler being told to link the header files to the system libraries.

Example

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

Definition Section

In this section, we define different constants. The keyword define is used in this part.

Example

```
#define PI=3.14
```

Global Declaration Section

This part of the code is the part where the global variables are declared. All the global variable used are declared in this part. The user-defined functions are also declared in this part of the code.

Example

```
float area(float r);

int a=7;
```

Main Function Section

Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

Example

```
int main(void)

{

int a=10;

printf(" %d", a);
return 0;

}
```

Sub Program Section

All the user-defined functions are defined in this section of the program.

Example

```
int add(int a, int b)

{

return a+b;

}
```

C Tokens

C Tokens are the smallest building block or smallest unit of a C program.
C Supports Six Types of Tokens:

- Identifiers
- Keywords
- Constants
- Strings
- Operators
- Special Symbols

Identifier

Identifiers are names given to different entities such as constants, variables, structures, functions, etc.
Eg:
int amount;

double totalbalance;
Rules:

1. An identifier can only have alphanumeric characters (a-z , A-Z , 0-9) (i.e. letters & digits) and underscore(_) symbol.
2. Identifier names must be unique
3. The first character must be an alphabet or underscore.
4. You cannot use a keyword as identifiers.
5. Only the first thirty-one (31) characters are significant.

6. It must not contain white spaces.
7. Identifiers are case-sensitive.

Keywords

Reserved words in C library and used to perform an internal operation. The meaning and working of these keywords are already known to the compiler.

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

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Constants

Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program.

1. Constants are also called literals.
2. Constants can be any of the data types.
3. It is considered best practice to define constants using only upper-case names.

Operators

C operators are symbols that are used to perform mathematical or logical manipulations. The C programming language is rich with built-in operators. Operators take part in a program for manipulating data and variables and form a part of the mathematical or logical expressions.
C programming language offers various types of operators having different functioning capabilities.

- Arithmetic Operators
- Relational Operators

- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Conditional Operator
- Bitwise Operators
- Special Operators

Arithmetic Operators

Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (*), division (/) and modulus (%).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Increment and Decrement Operators are useful operators generally used to minimize the calculation, i.e. ++x and x++ means x=x+1 or -x and x−−means x=x-1. But there is a slight difference between ++ or −− written before or after the operand. Applying the pre-increment first add one to the operand and then the result is assigned to the variable on the left whereas post-increment first assigns the value to the variable on the left and then increment the operand.

Operator	Description

++	Increment
−−	Decrement

Relational operators are used to comparing two quantities or values.

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical Operators

C provides three logical operators when we test more than one condition to make decisions. These are: && (meaning logical AND), || (meaning logical OR) and ! (meaning logical NOT).

Operator	Description

&&	And operator. It performs logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False)
||	Or operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True)
!	Not operator. It performs logical negation on an expression.

Bitwise Operators

C provides a special operator for bit operation between two variables.

Operator	Description
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator
~	Binary Ones Complement Operator
&	Binary AND Operator
^	Binary XOR Operator
|	Binary OR Operator

Assignment Operators

Assignment operators applied to assign the result of an expression to a variable. C has a collection of shorthand assignment operators.

Operator	Description
=	Assign
+=	Increments then assign
-=	Decrements then assign
*=	Multiplies then assign
/=	Divides then assign
%=	Modulus then assign
<<=	Left shift and assign
>>=	Right shift and assign
&=	Bitwise AND assign
^=	Bitwise exclusive OR and assign
|=	Bitwise inclusive OR and assign

Conditional Operator

C offers a ternary operator which is the conditional operator (?: in combination) to construct conditional expressions.

Operator	Description
? :	Conditional Expression

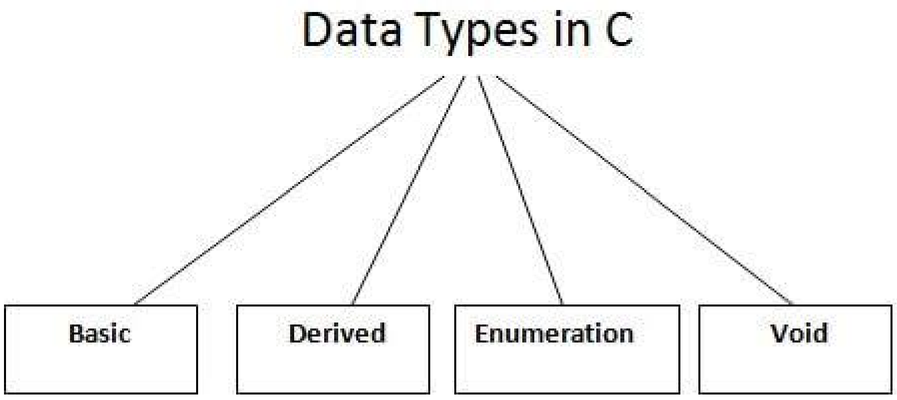
Special Operators

C supports some special operators

Operator	Description
sizeof()	Returns the size of a memory location.
&	Returns the address of a memory location.
*	Pointer to a variable.

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



There are the following data types in C language.

Types	**Data Types**
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.
The memory size of the basic data types may change according to 32 or 64-bit operating system.

Data Types	**Memory Size**	**Range**
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255

short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647

unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

CONTROL STRUCTURES IN C

C if else Statement

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true. There are the following variants of if statement in C language.

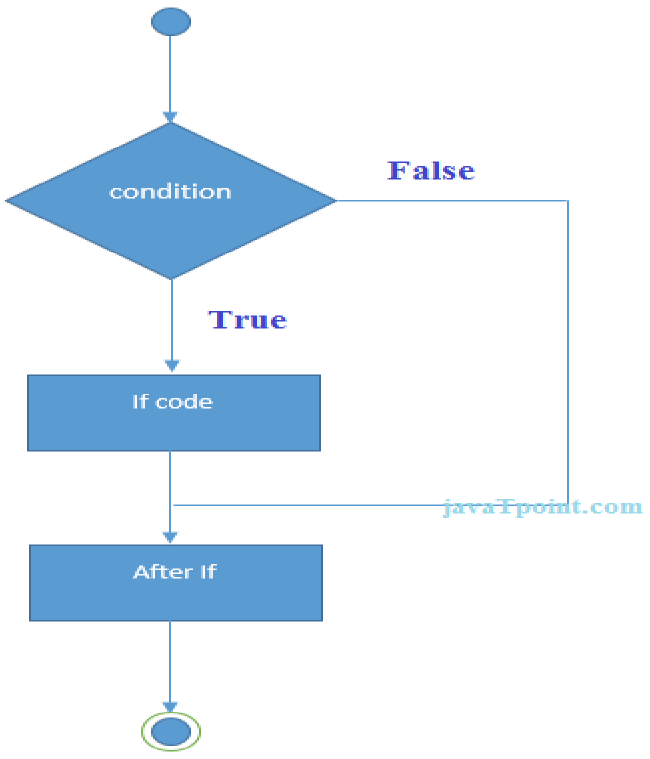
- If statement
- If-else statement
- If else-if ladder
- Nested if

If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

```
if(expression){
//code to be executed
}
```

Flowchart of if statement in C

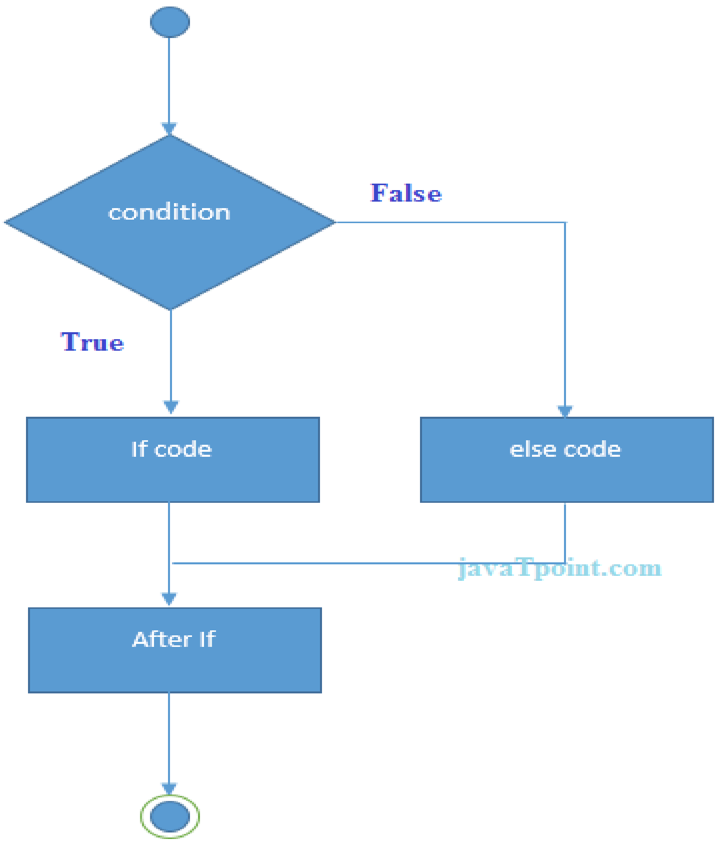


If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simultaneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

```
if(expression){
     //code to be executed if condition is true
          }else{
     //code to be executed if condition is false
               }
```

Flowchart of the if-else statement in C



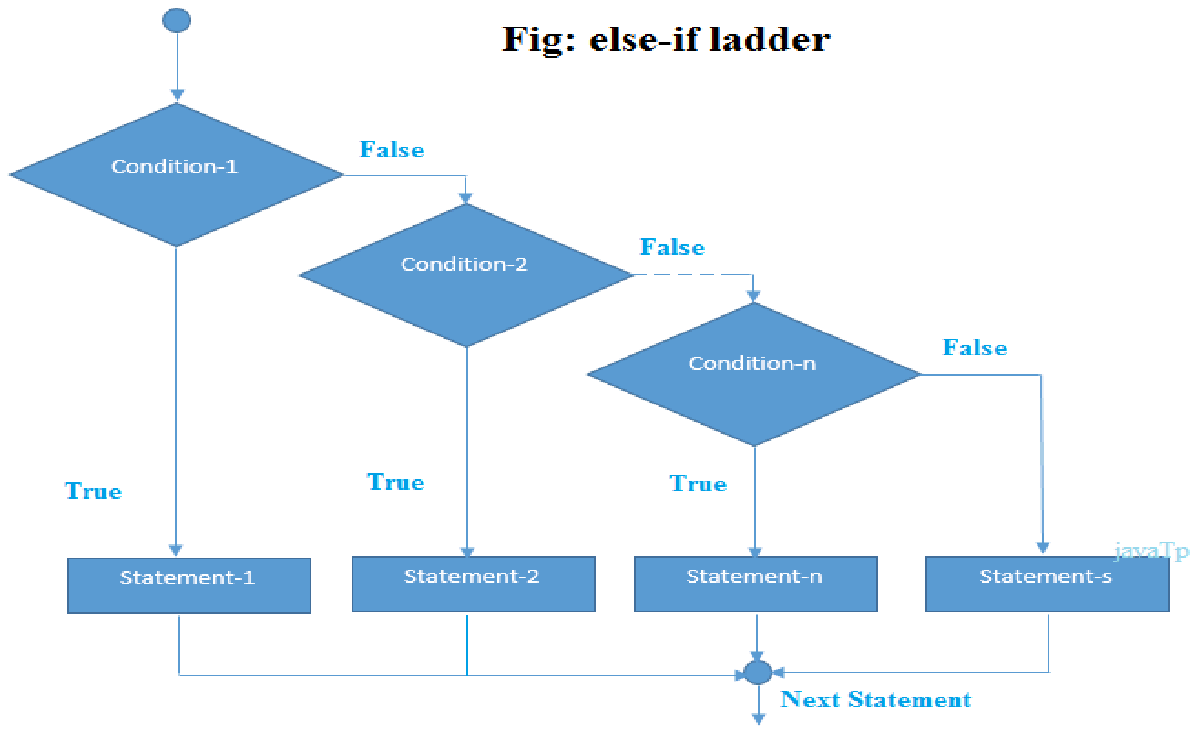
If else-if ladder Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1){
//code to be executed if condition1 is true
                    }else if(condition2){
                              //code to be executed if condition2 is true
                                             }else if(condition3){
                                                  //code to be executed if condition3 is true
                                                                      }
```

```
...
else{
        //code to be executed if all the conditions are false
    }
```

Flowchart of else-if ladder statement in C



C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

Syntax of switch statement in c language is given below:

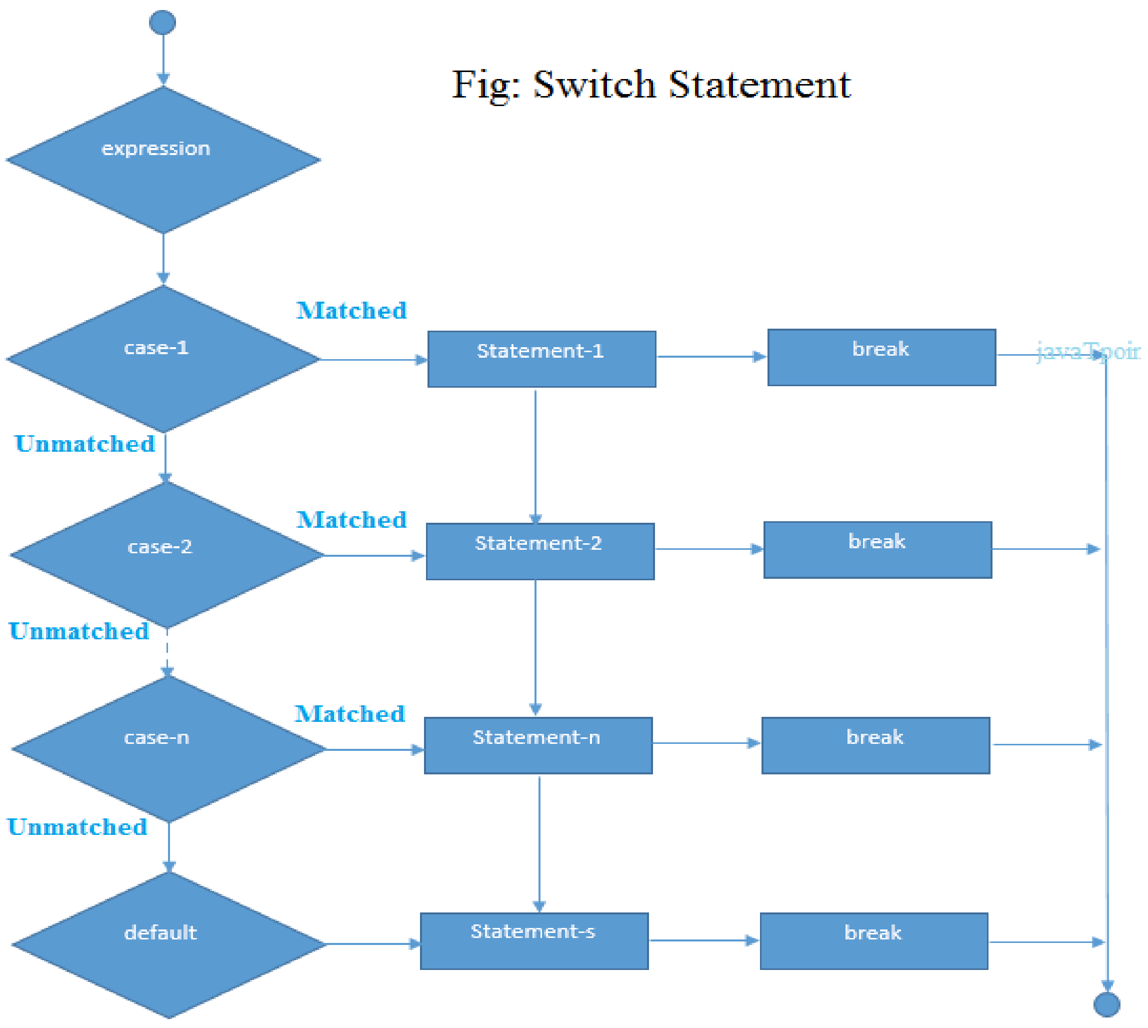
```
switch(expression){
        case value1:
        //code to be executed;
        break;  //optional
        case value2:
        //code to be executed;
        break;  //optional
        ......
```

```
    default:
 code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

1) The *switch expression* must be of an integer or character type.
2) The *case value* must be an integer or character constant.
3) The *case value* can be used only inside the switch statement.
4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Flowchart of switch statement in C



UNIT - II

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Advantage of loops in C

1) It provides code reusability.

2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while
2. while
3. for

do while loop in C

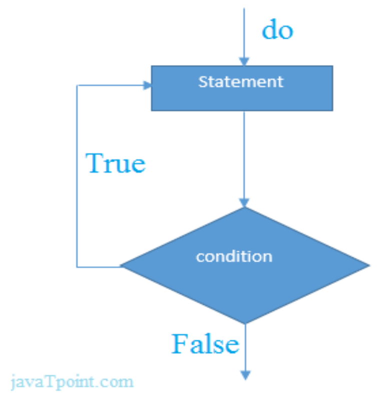
The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

do while loop syntax

The syntax of the C language do-while loop is given below:

```
do{
//code to be executed
}while(condition);
```

Flowchart of do while loop



Eg:#include<stdio.h>

```
int main(){
int i=1;
do{
printf("%d \n",i);
i++;
}while(i<=10);
return 0;
}
```

while loop in C

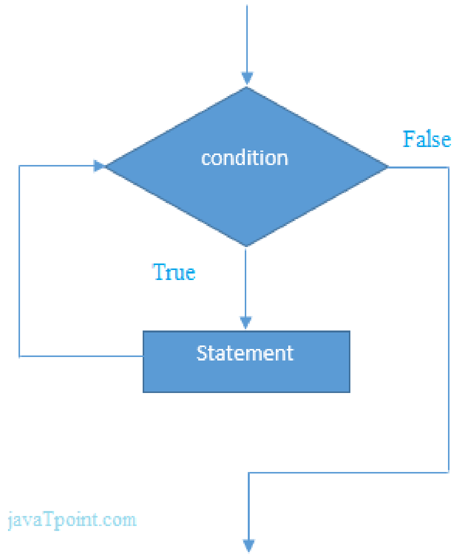
While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax of while loop in C language

The syntax of while loop in c language is given below:

```
while(condition){
//code to be executed
}
```

Flowchart of while loop in C



Example of the while loop in C language

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

for loop in C

The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

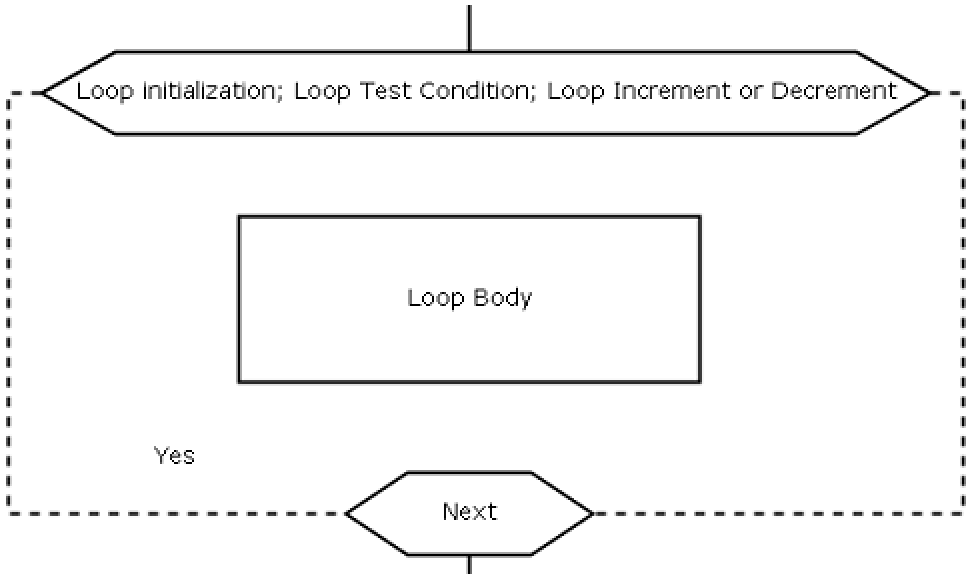
Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(Expression 1; Expression 2; Expression 3){
//code to be executed
```

}

Flowchart of for loop in C


Loop Initialization; Loop Test Condition; Loop Increment or Decrement
Loop Body
Yes
Next

C for loop Examples

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

C break statement

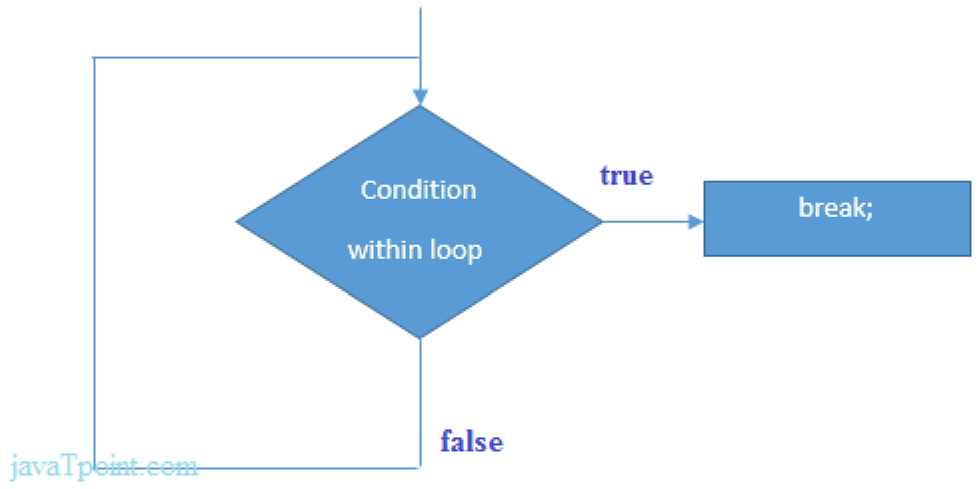
The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Flowchart of break in c



Example

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
   int i;
   for(i = 0; i<10; i++)
   {
      printf("%d ",i);
      if(i == 5)
      break;
   }
   printf("came outside of loop i = %d",i);

}
```

C continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

//loop statements

continue;

//some lines of the code which is to be skipped

Eg:#include<stdio.h>

```
void main ()
{
   int i = 0;
   while(i!=10)
   {
      printf("%d", i);
      continue;
      i++;
   }

}
```

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complecated.

Syntax:

label:

//some part of the code;

goto label;

goto example

#include <stdio.h>

```
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

ARRAYS

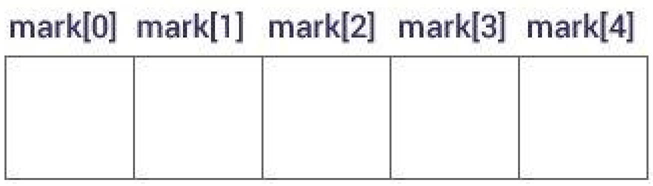
It uses only a single variable name for storing one data item. A new data structure is used called arrays.
(or)
In C language, arrays are referred to as structured data types. An array is defined as a finite ordered collection of homogenous data, stored in contiguous memory locations.

1. An array is a linear and homogeneous data structure
2. An array permits homogeneous data. It means that similar types of elements are stored contiguously in the memory under one variable name.
3. An array can be declared of any standard or custom data type.

Declaring an Array

Syn: data-type variable-name[size];
Eg: int mark[5];

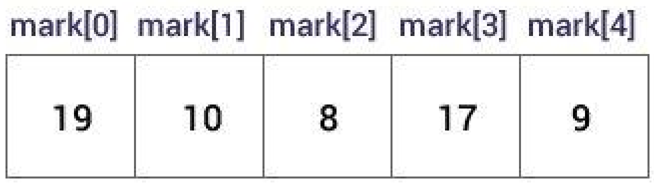
Pictorial representation of C Programming Arrays



Declare an Array

Initialization an Array

Eg: int mark[5]={19,10,8,17,9};



Initialize an Array

Types of Arrays

Single Dimensional Array

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array, data is stored in linear form. Single dimensional arrays are also called as **one-dimensional arrays**, **Linear Arrays** or simply **1-D Arrays**.

Declaration of Single Dimensional Array

General syntax for declaring a single dimensional array...

```
datatype arrayName [ size ] ;
```

Example Code

```
int rollNumbers [60] ;
```

The above declaration of single dimensional array reserves 60 contiguous memory locations of 2 bytes each with the name **rollNumbers** and tells the compiler to allow only integer values into those memory locations.

Initialization of Single Dimensional Array

The following general syntax for declaring and initializing a single dimensional array with size and initial values.

```
datatype arrayName [ size ] = {value1, value2, ...} ;
```

Example Code

```
int marks [6] = { 89, 90, 76, 78, 98, 86 } ;
```

Multi Dimensional Array
An array of arrays is called a multi dimensional array. In simple words, an array created with more than one dimension (size) is called a multi dimensional array. Multi dimensional array can be of **two dimensional array** or **three dimensional array** or **four dimensional array** or more...

Multi dimensional array is **two dimensional array**. The 2-D arrays are used to store data in the form of tables.

Declaration of Two Dimensional Array

We use the following general syntax for declaring a two dimensional array...

```
datatype arrayName [ rowSize ] [ columnSize ] ;
```

Example Code

```
int matrix_A [2][3] ;
```

The above declaration of two dimensional array reserves 6 contiguous memory locations of 2 bytes each in the form of **2 rows** and **3 columns**.

Initialization of Two Dimensional Array

The following general syntax for declaring and initializing a two dimensional array with a specific number of rows and columns with initial values.

```
datatype arrayName [rows][columns] = {{r1c1value, r1c2value, ...},{r2c1, r2c2,...}...} ;
```

Example Code

```
int matrix_A [2][3] = { {1, 2, 3},{4, 5, 6} } ;
```

The above declaration of two-dimensional array reserves 6 contiguous memory locations of 2 bytes each in the form of 2 rows and 3 columns. And the first row is initialized with values 1, 2 & 3 and second row is initialized with values 4, 5 & 6.

Example Code

```
int matrix_A [2][3] = { {1, 2, 3}, {4, 5, 6} } ;
```

UNIT - III

String in C

A String in C is a collection of characters in a linear sequence. ‘C’ always treats a string a single data even though it contains whitespaces. A single character is defined using single quote representation. A string is represented using double quote marks.
Example: "Welcome to the world of programming!"

‘C’ provides standard library <string.h> that contains many functions which can be used to perform complicated operations easily on Strings in C.

Declaration of strings: Declaring a string is as simple as declaring a one-dimensional array. Basic syntax for declaring a string.

datatype str_name[size];

In the above syntax str_name is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store which is the Null character (‘\0’) used to indicate the termination of string which differs strings from normal character arrays.

Initializing a String: A string can be initialized in different ways.

To declare a string with name as str and initialize it with “PVKNGC”.

1. char str[] = "PVKNGC";

2. char str[50] = "PVKNGC";

3. char str[] = {'P','V','K','N','G','C','\0'};

4. char str[14] = {'P','V','K','N','G','C','\0'};

C – String functions

1. STRLEN()**:** In C programming, we use strlen() to find the length of the string. String handling function strlen() returns the number of characters in string.

Syntax: integer_variable = strlen(string or string_variable);
Examples:

```
char str[30];

int len;
printf("Enter string:\n");
gets(str);
len = strlen(str);
 printf("Length of given string is: %d", len);
```

2. STRCPY(): String handling function strcpy() is used to copy content of one string variable to another string variable i.e. strcpy(string2, string1) copies content of string1 to string2.

Syntax: integer_variable = strcpy (string2, string1);
Examples:

```
char str1[30], str2[30];


printf("Enter string:\n");
gets(str1);
strcpy(str2, str1);
printf("Coped string is: %s", str2);
```

3. STRCMP(): In C programming, string handling function strcmp() is used to compare two strings. This function returns 0 if two strings are same otherwise it returns some integer value other than 0.

Syntax: integer_variable = strcmp(string1, string2);

Examples:

```
char str1[40], str2[40];
 int d;
printf("Enter first string:\n");
gets(str1);
 printf("Enter second string:\n");
```

```
gets(str2);
d = strcmp(str1, str2);
if(d==0)
{
 printf("Given strings are same.");
}
else
{
 printf("Given strings are different.");
}
```

4. STRCAT():String handling function strcat() is used to concatenate two strings. Concatenation is the process of merging content of one string to another string. Function strcat(str1, str2) concatenates content of string str2 after content of string str1.

Syntax: integer_variable = strcat(string1, string2);

Examples:

```
char str1[50], str2[50];

printf("Enter first string:\n");

gets(str1);

printf("Enter second string:\n");

gets(str2);

strcat(str1,str2);

printf("Concatenated string is: %s", str1);
```

5. STRREV():String handling function strrev() is used to reverse string in C programming language. Function strrev(str1) reverses the content of string str1.

Syntax: integer_variable =strrev(string);
Examples:

```
char name[40];
```

```
printf("Enter your name: ");

gets(name);

strrev(name);

printf("Reversed name is: %s", name);
```

6. STRUPR():String handling function strupr() is used to convert all lower case letter in string to upper case i.e. strlwr(str) converts all lower case letter in string **str** to upper case.

Syntax: integer_variable =strupr(string);

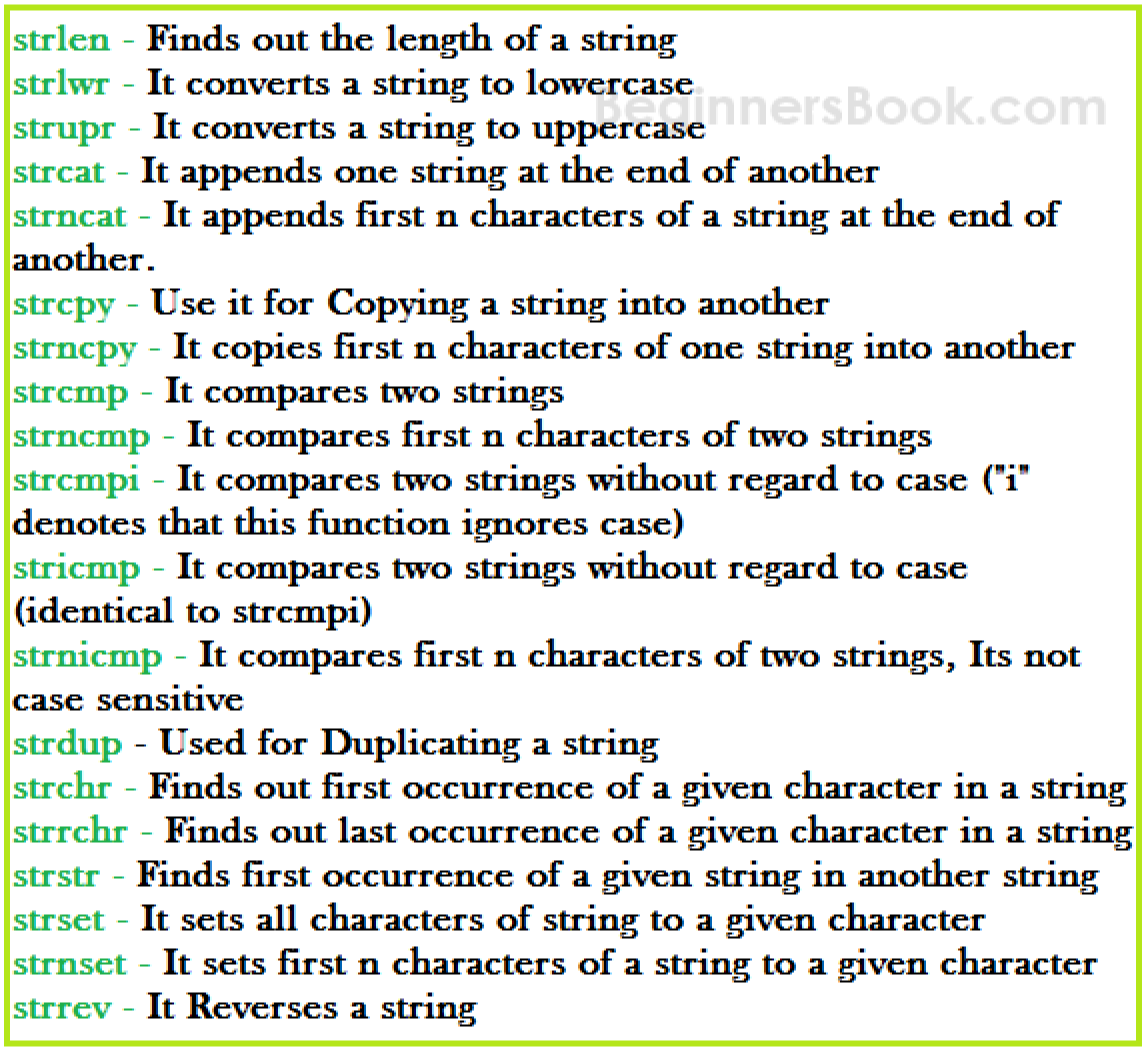
Examples:

```
char str[40];

 printf("Enter string:\n");

 gets(str);

 strupr(str);

 printf("String in uppercase is:");

 puts(str);
```

7. STRLWR():String handling function strlwr() is used to convert all upper case letter in string to lower case i.e. strlwr(str) converts all upper case letter in string **str** to lowercase.

Syntax: integer_variable =strlwr(string);
Examples:

```
char str[40];

 printf("Enter string:\n");

 gets(str);

 strlwr(str);

 printf("String in lowercase is:");

 puts(str);
```


strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

C Functions

In c, a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure or subroutine in* other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- Function calling is always overhead in a C program.

Function Aspects

There are three aspects of a C function.

- **Function declaration:** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call:** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. Pass the same number of functions as it is declared in the function declaration.
- **Function definition:** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called, only one value can be returned from the function.

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list){function body;}

The syntax of creating function in c language is given below:

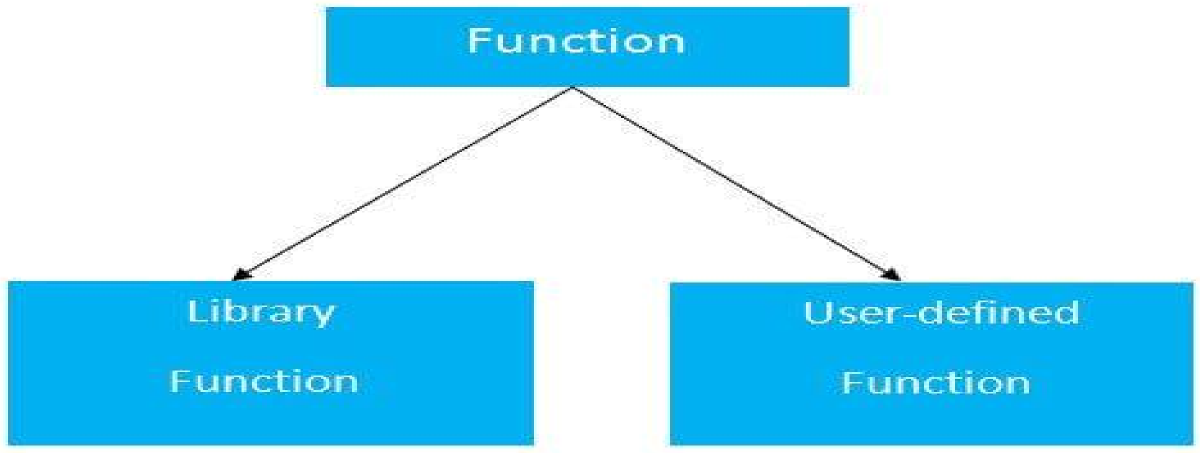
1. return_type function_name(data_type parameter...){

2. //code to be executed
3. }

Types of Functions

There are two types of functions in C programming:

1. **Library Functions** are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions** are the functions which are created by the C programmer, so that they can use it many times. It reduces the complexity of a big program and optimizes the code.



Return Value

A C function may or may not return a value from the function. To return any value from the function, use void for the return type.

Example without return value:

1. void hello(){
2. printf("hello c");
3. }

To return any value from the function, use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Example with return value:

```
int get(){
return 10;
}
```

In the above example, we have to return 10 as a value, so the return type is int. To return floating-point value (e.g., 10.2, 3.1, 54.5, etc), use float as the return type of the method.

```
float get(){
return 10.2;
}
```

Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

Example for Function without argument and return value

Example 1

```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```

Example 2

```
#include<stdio.h>
void sum();
void main()
{
    printf("\nGoing to calculate the sum of two numbers:");
    sum();
}
void sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}
```

Example for Function without argument and with return value

Example 1

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

Example 2: program to calculate the area of the square

```
#include<stdio.h>
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    float area = square();
    printf("The area of the square: %f\n",area);
}
int square()
{
    float side;
    printf("Enter the length of the side in meters: ");
    scanf("%f",&side);
    return side * side;
}
```

Example for Function with argument and without return value

Example 1

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

Example 2: program to calculate the average of five numbers.

```
#include<stdio.h>
void average(int, int, int, int, int);
void main()
{
    int a,b,c,d,e;
    printf("\nGoing to calculate the average of five numbers:");
    printf("\nEnter five numbers:");
    scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
    average(a,b,c,d,e);
}
void average(int a, int b, int c, int d, int e)
```

```
{
    float avg;
    avg = (a+b+c+d+e)/5;
    printf("The average of given five numbers : %f",avg);
}
```

Example for Function with argument and with return value

Example 1

```
#include<stdio.h>
int sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    result = sum(a,b);
    printf("\nThe sum is : %d",result);
}
int sum(int a, int b)
{
    return a+b;
}
```

Example 2: Program to check whether a number is even or odd

```
#include<stdio.h>
int even_odd(int);
void main()
{
 int n,flag=0;
 printf("\nGoing to check whether a number is even or odd");
 printf("\nEnter the number: ");
 scanf("%d",&n);
 flag = even_odd(n);
 if(flag == 0)
 {
   printf("\nThe number is odd");
 }
 else
 {
   printf("\nThe number is even");
 }
}
int even_odd(int n)
{
   if(n%2 == 0)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
```

C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension .h. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called a recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls.

Recursion cannot be applied to all the problems, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function calls are always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```

UNIT - IV

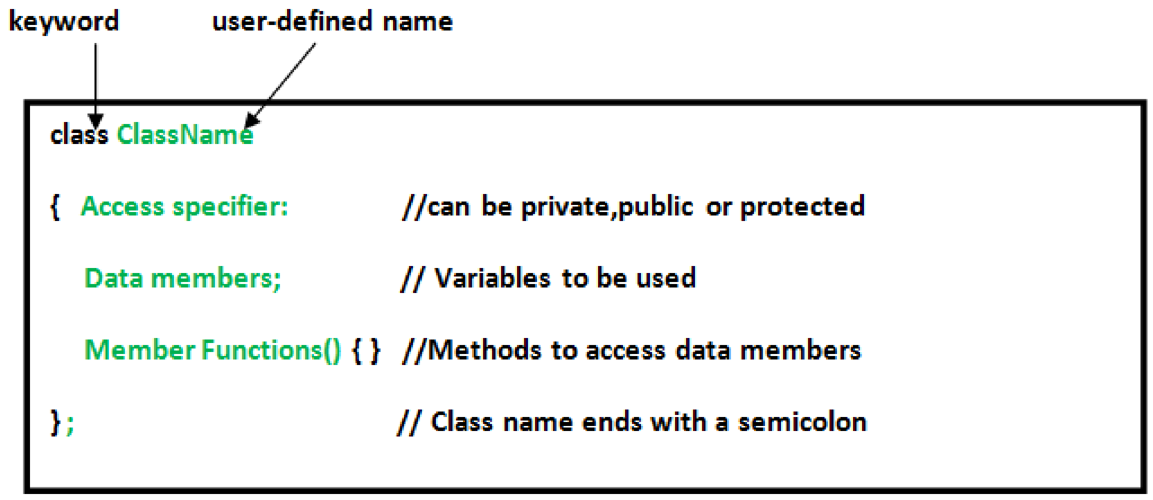
C++ Classes and Objects

Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName;

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : **public, private and protected**.

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

Array of Objects in C++

Array of other user-defined data types, an array of type class can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

The syntax for declaring an array of objects is

```
class_name array_name [size] ;
```

Example:

```
#include<iostream>
using namespace std;
class books
{
char tit1e [30];
float price ;
public:
```

```
void getdata ();
void putdata ();
} ;
void books :: getdata ()
{
cout<<"Title:”;
Cin>>title;
cout<<"Price:”;
cin>>price;

}
void books :: putdata ()
{
cout<<"Title:"<<title<< "\n";
cout<<"Price:"<<price<< "\n”;
const int size=3 ;
int main ()
{
books book[size] ;
for(int i=0;i<size;i++)
{
cout<<"Enter details o£ book "<<(i+1)<<"\n";
book[i].getdata();
}
for(int i=0;i<size;i++)
{
cout<<"\nBook "<<(i+l)<<"\n";
book[i].putdata() ;
}
return 0;
}
```

C++ Friend Function

Private members are accessed only within the class they are declared. Friend function is used to access the private and protected members of different classes. It works as bridge between classes.

- Friend function must be declared with **friend** keyword.
- Friend function must be declare in all the classes from which we need to access private or protected members.
- Friend function will be defined outside the class without specifying the class name.
- Friend function will be invoked like normal function, without any object.

```
#include<iostream.h>
        class RectangleTwo;

        class RectangleOne
        {
                int L,B;
                public:
                RectangleOne(int l,int b)
                {
                L = l;
                B = b;
                }
                friend void Sum(RectangleOne, RectangleTwo);
        };
        class RectangleTwo
        {
                int L,B;
                public:
                RectangleTwo(int l,int b)
                {
                L = l;
                B = b;
                }
                friend void Sum(RectangleOne, RectangleTwo);
        };
        void Sum(RectangleOne R1,RectangleTwo R2)
        {
                cout<<"\n\t\tLength\tBreadth";
                cout<<"\n Rectangle 1  :   "<<R1.L<<"\t  "<<R1.B;
                cout<<"\n Rectangle 2  :   "<<R2.L<<"\t  "<<R2.B;
                cout<<"\n -------------------------------";
                cout<<"\n\tSum   :   "<<R1.L+R2.L<<"\t  "<<R1.B+R2.B;
                cout<<"\n -------------------------------";
        }


        void main()
        {
                RectangleOne Rec1(5,3);
                RectangleTwo Rec2(2,6);
                Sum(Rec1,Rec2);
        }
```

Friend Class

A class can also be a friend of another class. A friend class can access all the private and protected members of other class.

To access the private and protected members of a class into friend class we must pass on object of a class to the member functions of friend class.

```
#include<iostream.h>
        class Rectangle
        {
                int L,B;
                public:
                Rectangle()
                {
                        L=10;
                        B=20;
                }
                friend class Square;        //Statement 1
        };
        class Square
        {
                int S;
                public:
```

```
                Square()
                {
                        S=5;
                }
                void Display(Rectangle Rect)
                {
                        cout<<"\n\n\tLength : "<<Rect.L;
                        cout<<"\n\n\tBreadth : "<<Rect.B;
                        cout<<"\n\n\tSide : "<<S;
                }
        };
        void main()
        {
                Rectangle R;
                Square S;
                S.Display(R);       //Statement 2

        }
```

Constructor

Constructor is a special function used to initialize class data members or we can say constructor is used to initialize the object of class.

Characteristics constructor.

- Constructor name class name must be the same.
- Constructor doesn't return value.
- Constructor is invoked automatically, when the object of the class is created.

Types of Constructor

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructor

Constructing without parameters is called the default constructor.

Example of C++ default constructor

```
        #include<iostream.h>
        #include<string.h>
        class Student
        {

                        int Roll;
                        char Name[25];
                        float Marks;
                        public:
                        Student()         //Default Constructor
                        {
                            Roll = 1;
                            strcpy(Name,"Kumar");
                            Marks = 78.42;
                        }
                        void Display()
                        {
                            cout<<"\n\tRoll : "<<Roll;
                            cout<<"\n\tName : "<<Name;
                            cout<<"\n\tMarks : "<<Marks;
                        }
        };
        void main()
        {
                        Student S;        //Creating Object
                        S.Display();      //Displaying Student Details
        }
```

Parameterized Constructor

Constructing with parameters is called a parameterized constructor.

Example of C++ parameterized constructor

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student
{
        int Roll;
        char Name[25];
        float Marks;
        public:
        Student(int r,char nm[],float m)        //Parameterized Constructor
        {
                Roll = r;
                strcpy(Name,nm);
                Marks = m;
        }
        void Display()
        {
                cout<<"\n\tRoll : "<<Roll;
                cout<<"\n\tName : "<<Name;
                cout<<"\n\tMarks : "<<Marks;
        }
};
void main()
{
        Student S(2,"Sumit",89.63);
        //Creating Object and passing values to Constructor
        S.Display();
        //Displaying Student Details
}
```

Copy Constructor

Initialization of an object through another object is called **copy constructor**. In other words, copying the values of one object into another object is called a copy **constructor.**

Example of C++ copy constructor

```
	#include<iostream.h>
	#include<conio.h>
	#include<string.h>
	class Student
	{
		int Roll;
		char Name[25];
		float Marks;
		public:
		Student(int r,char nm[],float m)   //Constructor 1 : Parameterized Constructor
		{
			Roll = r;
			strcpy(Name,nm);
			Marks = m;
		}
		Student(Student &S)   //Constructor 2 : Copy Constructor
		{
			Roll = S.Roll;
			strcpy(Name,S.Name);
			Marks = S.Marks;
		}
		void Display()
		{
			cout<<"\n\tRoll : "<<Roll;
			cout<<"\n\tName : "<<Name;
			cout<<"\n\tMarks : "<<Marks;
```

```
                }
        };
        void main()
        {
                Student S1(2,"Sumit",89.63);
                Student S2(S1);    //Statement 1
                cout<<"\n\tValues in object S1";
                S1.Display();
                cout<<"\n\tValues in object S2";
                S2.Display();
        }
```

Destructor

Constructor allocates the memory for an object.

Destructor deallocate the memory occupied by an object. Like constructor, destructor name and class name must be the same, preceded by a tilde(~) sign. Destructor take no argument and have no return value.

Constructor is invoked automatically when the object is created.

Destructor is invoked when the object goes out of scope. In other words, Destructor is invoked, when the compiler comes out from the function where an object is created.

Example of C++ destructor

```
        #include<iostream.h>
        #include<conio.h>
        #include<string.h>
        class Student
        {
                int Roll;
                char Name[25];
                float Marks;
                public:
```

```
        Student()   //Default Constructor
        {
                Roll = 4;
                strcpy(Name,"Sumit");
                Marks = 84.56;
        }
        void Display()
        {
                cout<<"\n\tRoll : "<<Roll;
                cout<<"\n\tName : "<<Name;
                cout<<"\n\tMarks : "<<Marks;
        }
        ~Student()   //Destructor
        {
                cout<<"\n\tEnd of program.";
        }
    };
    void main()
    {
        Student S;
        S.Display();
    }
```

Constructor Overloading

In C++, Constructor is automatically called when an object(an instance of the class) is created. It is the special member function of the class.Which constructor has arguments is called Parameterized Constructor.

- One Constructor overload another constructor is called Constructor Overloading
- It has the same name as the class.
- It must be a public member.

- No Return Values.
- Default constructors are called when constructors are not defined for the classes.

Syntax

```
class class_name
{
Access Specifier : Member_Variables Member_Functions
public: class_name()
{ // Constructor code
} class_name(variables)
      { // Constructor code
      } ... other Variables & Functions
     }
    #include<iostream>
    #include<conio.h>
    class Example
     { // Variable Declaration
    int a, b;
    public: //Constructor without Argument
    Example() { // Assign Values In Constructor
    a = 50; b = 100;
    cout << "\nIm Constructor";
    } //Constructor with Argument
    Example(int x, int y)
     { // Assign Values In Constructor
     a = x; b = y;
    cout << "\nIm Constructor";
    } void Display()
    { cout << "\nValues :" << a << "\t" << b; } };
    int main()
    {
     Example Object(10, 20);
    Example Object2; // Constructor invoked.
    Object.Display();
    Object2.Display(); // Wait For Output Screen
    getch();
}
```

UNIT - V

Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

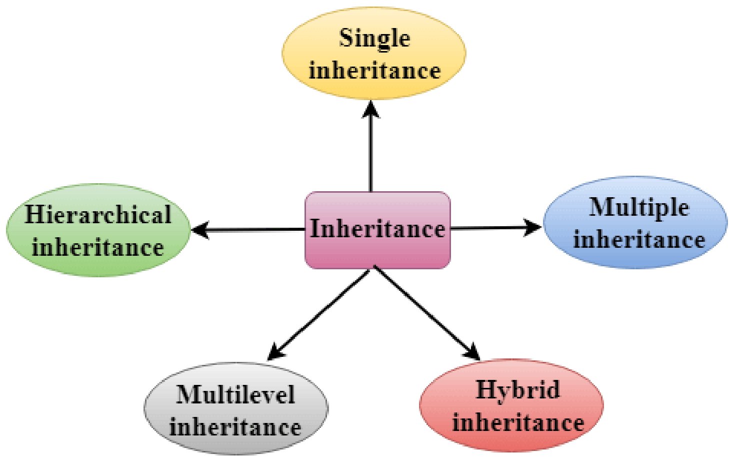
Advantage of C++ Inheritance

Code reusability: Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

Types Of Inheritance

C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



Derived Classes

A Derived class is defined as the class derived from the base class.

The Syntax of Derived class:

```
class derived_class_name :: visibility-mode base_class_name
{
    // body of the derived class.
}
```

Where,

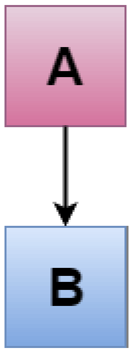
derived_class_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

base_class_name: It is the name of the base class.

C++ Single Inheritance

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.



Where 'A' is the base class, and 'B' is the derived class.

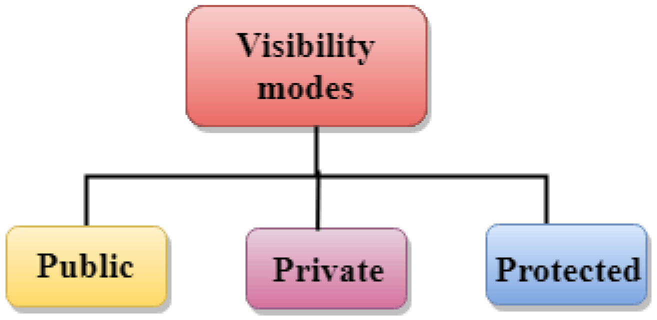
Example

```
#include <iostream>
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
```

```
  };
int main(void) {
    Programmer p1;
    cout<<"Salary: "<<p1.salary<<endl;
    cout<<"Bonus: "<<p1.bonus<<endl;
   return 0;
}
```

C++ introduces a third visibility modifier, i.e., **protected**. The member which is declared as protected will be accessible to all the member functions within the class as well as the class immediately derived from it.

Visibility modes can be classified into three categories:



- **Public**: When the member is declared as public, it is accessible to all the functions of the program.
- **Private**: When the member is declared as private, it is accessible within the class only.
- **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

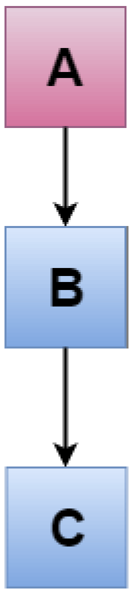
Visibility of Inherited Members

Base class visibility	**Derived class visibility**		
	Public	**Private**	**Protected**

Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

C++ Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.

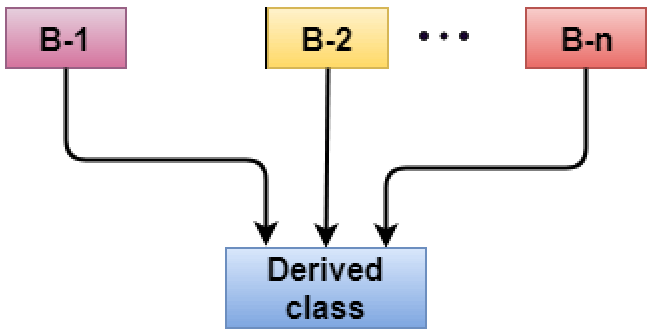


Example

```
#include <iostream>
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
       public:
     void bark(){
    cout<<"Barking..."<<endl;
     }
   };
   class BabyDog: public Dog
   {
       public:
     void weep() {
    cout<<"Weeping...";
     }
   };
int main(void) {
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

C++ Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



Syntax of the Derived class:

```
1. class D : visibility B-1, visibility B-2, ?
2.  {
```

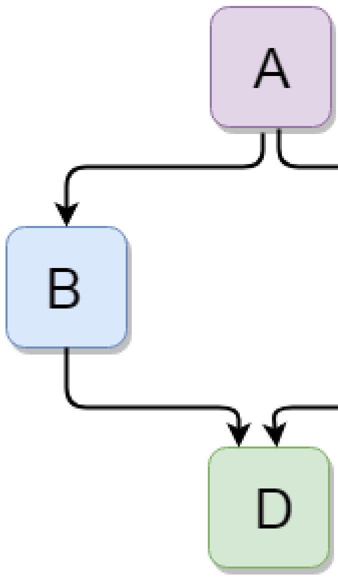
3. // Body of the class;
4. }

Example

```
#include <iostream>
class A
{
   protected:
    int a;
   public:
   void get_a(int n)
   {
      a = n;
   }
};

class B
{
   protected:
   int b;
   public:
   void get_b(int n)
   {
      b = n;
   }
};
class C : public A,public B
{
  public:
   void display()
   {
      std::cout << "The value of a is : " <<a<< std::endl;
      std::cout << "The value of b is : " <<b<< std::endl;
      cout<<"Addition of a and b is : "<<a+b;
   }
};
int main()
{
  C c;
  c.get_a(10);
  c.get_b(20);
  c.display();

   return 0;
}
```

C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.

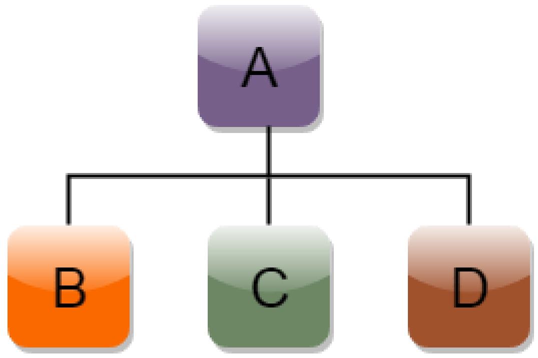


Example:

```
#include <iostream>
class A
{
    protected:
    int a;
    public:
    void get_a()
    {
       std::cout << "Enter the value of 'a' : " << std::endl;
       cin>>a;
    }
};

class B : public A
{
    protected:
    int b;
    public:
    void get_b()
    {
        std::cout << "Enter the value of 'b' : " << std::endl;
       cin>>b;
    }
};
class C
{
    protected:
    int c;
    public:
    void get_c()
    {
        std::cout << "Enter the value of c is : " << std::endl;
        cin>>c;
    }
};

class D : public B, public C
{
    protected:
    int d;
    public:
    void mul()
    {
         get_a();
         get_b();
         get_c();
         std::cout << "Multiplication of a,b,c is : " <<a*b*c<< std::endl;
    }
};
int main()
{
    D d;
    d.mul();
    return 0;
}
```

C++ Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Syntax of Hierarchical inheritance:

```
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Example:

```
#include <iostream>

class Shape                  // Declaration of base class.

{

   public:

   int a;
```

```
    int b;
    void get_data(int n,int m)
    {
        a= n;
        b = m;
    }
};
class Rectangle : public Shape  // inheriting Shape class
{
    public:
    int rect_area()
    {
        int result = a*b;
        return result;
    }
};
class Triangle : public Shape    // inheriting Shape class
{
    public:
    int triangle_area()
    {
        float result = 0.5*a*b;
        return result;
    }
};
int main()
```

```
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    std::cout << "Area of the rectangle is : " <<m<< std::endl;
    std::cout << "Enter the base and height of the triangle: " << std::endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    std::cout <<"Area of the triangle is : "  << n<<std::endl;
    return 0;
}
```