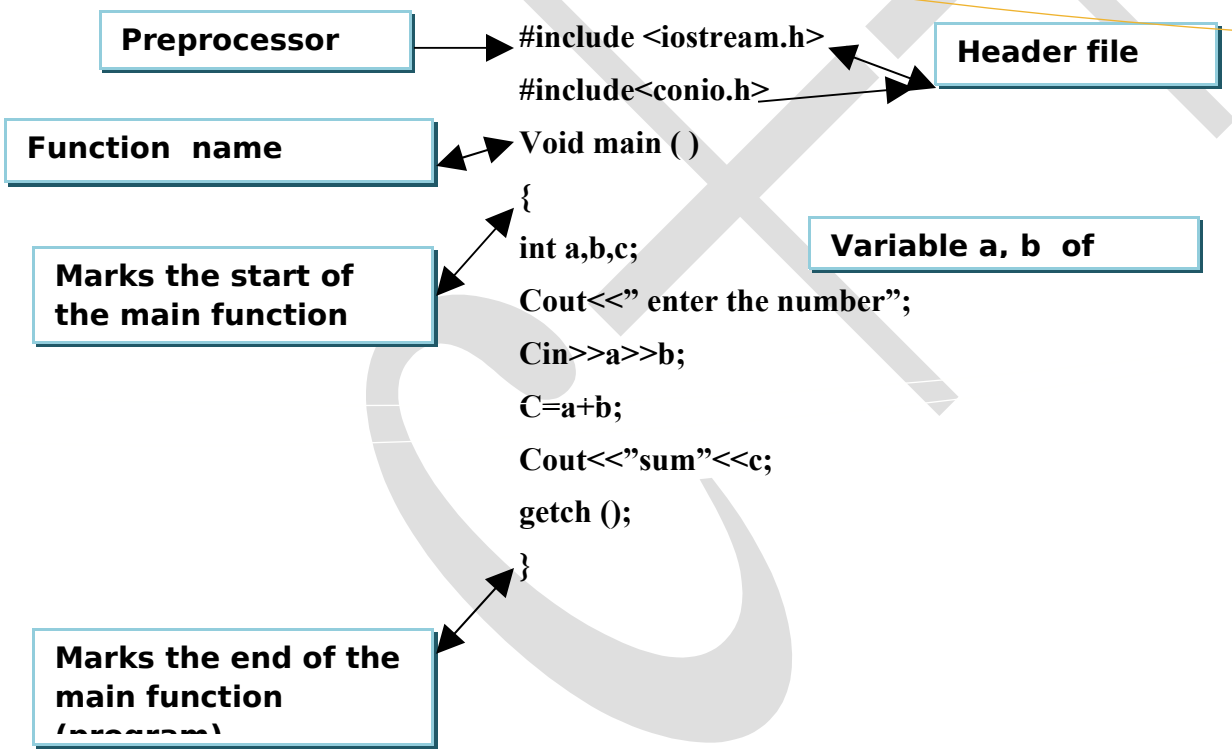
# TUTORIALS OF C++

## The history of C++:

C++ was developed by Bjarne Stroustrup of AT&T Bell Laboratories in the early 1980's, and is based on the C language. C++ is an object oriented programming language, it implements “data abstraction “using a concept called “classes” along with some other features of oop,apart of the c++ program are easily reasuble and extensible code is easily modifiable withoue actually having to change the code . The "++" is a syntactic construct used in C (to increment a variable), and C++ is intended as an incremental improvement of C .It contains all features of oops. A simple program:- A simple program for the addition of two numbers-




**For input - cin** **<< : insertion symbol ,**

**For output – cout** **; : semicolon**

**Input /output stream- iostream.h** **“ “ : double quote**

**Console input/output- conio.h** **( ) : parenthesis**

***What is meant by object-oriented programming?***

**Ans**: OOPs is the new concept of programming parallel to Procedure oriented programming.It were intorduced in late 80's.It consider the programming simulated to real world objects.It help in programming approach in order to built robust user friendly and efficient softwares and provide the efficient way to maintain real world softwares. OOPs is an Object Oriented Programming language which is the extension of Procedure Oriented Programming language.OOps reduce the code of the program because of the extensive feature of Polymorphism.OOps have many properties such as DataHiding Inheritence Data Absraction Data Encapsulation and many more. Opps is Object oriented programming language.The main aim is to creating an Object to The Entire program and that to we can control entire program using the Object.the main features of OPPS is Polymorphism,Multiple Inheritence,abstraction and encapsulation.

## Basic concepts of oops:

The different concepts of OOPs are as follows

**(a) Encapsulation:** It is used to hide the data as well as the binding of a data members and member functions.

**b) Inheritance:** It is the process by which one class inherits the properties of another Class.

**(c) Polymorphism:** poly means many and morphs means form, so polymorphism Means one name multiple form.there are two types of polymorphism :compile time and run time polymorphism.

**(d) Data hiding:** This is the property in which some of the members are restricted from Outside access. This is implemented by using private and protected access specifies.

**(e) Data abstraction:** This is the property in which only necessary information is Extracted. This property is implemented by using the class in C++.

(f) **Class:** It is a user defines data type which contains data member & member function. It is collection of various kind of object. It is define by class keyword. It also an Important feature of object oriented programming language. For ex-fruit is a class And apple, mango, banana are its object.

(g) **Objec**t: An object is a basic run time entity. Object represents/resembles a Physical/real entity. An object is simply something you can give a name.

All the objects have some characteristics and behaviour.the state ofn an object represent all the information held within it and behavior of an object is the set of action that it can perform to change the state of the object.

All real world object have three characterstics:

- State : How object react?
- Behaviour : what we can do with this object?

- Identity : difference between one object to another object?

For ex- our bike

- State : (gear,speed,fuel)
- Behaviour : (changing speed ,applying brakes)
- Identity : (registration number,engine number)

(h) **Overloading**: Adding a new method with the same name in same/derived class but With different number/types of parameters. It implements Polymorphism.

**Write the merits and demerits of object-oriented language as compared to procedure-oriented language.**

**Ans:** We can compare the procedure-oriented programming(c) with the object-oriented (c++) .

- pop focus more on function
- oop focus on data
- oop deals with real world object
- In pop error detection is difficult as we can't know which Variable is associated with which function
- In oop we can specify with the object that which variable is Associated with which function
- objects in oop creates many modules of program which is Flexible and easier to execute and also understand
- OOP provides inheritance in which features can be added to Existing classes without modification

**a]**In pop importance is given for doing things**.** In oop importance is given on data rather than procedure.

**b]**Pop, most of function share global data**.** oop,data structure are designed such that the characteristics the object function that operate on the data of an object which are tied together in the data structure

**c]** Pop, larger programs are divided into smaller programs known as function**.**oop,program are divided into smaller programs known as objects.

**d]**in pop security is not provided for object**.** In oop security is provided to data.

**e]**in pop top down approach**.** In oop bottom up approach.

**Tokens** - The smallest individual units in a program are known as tokens, c++ has the following tokens:
Example of tokens:- } ,{, " ", int

- Keywords
- Identifiers
- Constants

**Identifier:**
In our everyday, we give names to different things so they can be
Referred easily. Similarly, in C+, we use identifiers to name user created entities
Which may be?

- Variable
- Function
- Type e.g. a class

Every thing has some restrictions and exceptions along with many permissible things. So, does C++ by putting some restrictions on how we can name these entities. Let us see these rules in details:

1. An identifier can be combination of letters, numbers, and underscores with following restrictions:

a) It should start with a letter or underscore. E.g. height, my_height,_myHeight are allowed but not 1isGod
b) If it starts with a underscore then the first letter should not be capital because such names are reserved for implementation. E.g. _Height not allowed

2. It should be unique in a program taking care that C++ is case sensitive. E.g. age and Age are different variables

3. A keyword cannot be used as an identifier.

4. There is no restriction on length of the identifier. E.g. h and h_represents_my height are both valid.

Besides restrictions, there are certain guidelines which you should follow:

**a.)** Use meaningful descriptive names. E.g. int Age is better than int a.
- If description makes identifier name too long then put a comment before identifier and make identifier shorter
**b).** Be consistent in your naming convention.
- Use small letters for single word identifier name.
- For multiword identifiers, either use underscore separated or intercepted notation. E.g. get_my_height () or getMyHeight ()

**c.)** Use Hungarian notation. E.g. double dFlowRate, int value, bool check.
**d.)** Don't use similar names in a program like Speed, speed, and Speedy
**e.)** Don't use capitalized version of a keyword like Return

## Keywords:

Keywords are predefined reserved identifiers that have special meanings. They cannot be used as identifiers in your program.

*Keyword is a word that the compiler already knows*, i.e. when the compiler sees a keyword somewhere in the program it knows what to do automatically.

*For example*, when the compiler encounters the keyword 'int', it knows that 'int' stands for an integer. Or if the compiler reads a 'break', then it knows that it should break out of the current loop. Some common keywords are-

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

**Constant :**

As the name suggests, a variable is something whose value can be changed throughout the program. It is not fixed. On the other hand, a constant is one whose value remains the same (constant) throughout the program.

- **Variable:** A variable is the storage location in memory that is stored by its value. A variable is id denoted by a variable name. The variable name is a sequence of one or more letters, digits or und

- **Variable decleration**

declaration : int a;
declaration means here a is declared as integer variable

**Declaring and defining variables**
**A variable in C++ must be declared (the type of variable) and defined (values assigned to a variable) before it can be used in a program. Following shows how to declare a variable.**

**Rules of variable declaration**

- **A variable name can have one or more letters or digits or underscore, for example character _.**
- **White space, punctuation symbols or other characters are not permitted to denote variable name. .**
- **A variable name must begin with a letter.**
- **Variable names cannot be keywords or any reserved words of the C++ programming language.**
- **C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with written in small letters.**
- **For example, the variable name CIST differs from the variable name cist.**

## Variable Definition vs Declaration

| | |
|---|---|
| **Definition**<br><br>**Ex - int a=5** | Tell the compiler about the variable: its type and name, as well as allocated a memory cell for the variable |
| **Declaration**<br><br>**Ex- int a** | Describe information ``about'' the variable, doesn't allocate memory cell for the variable |

**Operators:** operators play a great role in any languages, the operations are represented by operators and the objects of the operation are referred to as operands. There are four types of operators.

- Arithmetic
- Relational
- Logical
- Bitwise

In addition, there are some special operators for special tasks.
Operator can be **unary (involve 1 operand) , binary(involve 2 operands),and ternary(involve 3 operands)**.

### ➢ Arithmetic operator

In any language, there are some operators to perform arithmetic, logical and control operations.The basic operators which are used to perform arithmetic operations on integers are as follows:-

| Operator | Operation |
|---|---|
| + | Addition, also called unary addition |
| - | Subtraction, also called unary subtraction |
| * | Multiplication |
| / | Division |

| | |
|---|---|
| % | Modulus (remainder after division) |
| ++ | Increment |
| -- | Decrement |

The operators +,-,* and / perform the same operation as they do in other languages. The operators + and – are unary operators and can take one or two operands. The multiply and divide are called binary operators as they take two operands.

Integer division will always give the result which is of integer type and truncates the remainder. The output of integer division will be the quotient. For example, 5/2 will give the answer 2 and not 2.5. The modulus operator gives the remainder after the integer division. For example, 5/2 will give the answer 1, which is the remainder.

Example-

```
# include<iostream.h>
#include<conio.h>
Void main( )
{
Clrscr ( );
int  a=20,b=30,d,e,f,g;
d=a+b;
e=a-b;
f=a*b;
g=a%b;
cout<<d<<endl;
cout<<e<<endl;
cout<<f<<endl;
cout<<g<<endl;
getch( );
}
```

Result/output:
d=50, e=-10, f=600, g=0.

- **Relational operator**: The relational operator, refer to the relationship that value can have with one another, it use the Boolean values contains true or false.0 means false and 1 means true.

| Operator | Operation |
|---|---|
| > | Greater than |
| >= | Greater than equal to |

| | |
|---|---|
| < | Less than |
| <= | Less than equal to |
| == | Equal to |
| != | Not equal to |

- **Logical operator :**

| **Operator** | **Operation** |
|---|---|
| && | And |
| \|\| | OR |
| ! | NOT |

**Table for && (And operator):** The operator && corresponds with Boolean logical operation *AND*. This operator returns the value of true if both its operands are true or if it returns false. The following table reflects the value of && operator

| p | q | P&&q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| p | q | P&&q |
|---|---|---|
| F | F | F |
| F | T | F |
| T | F | F |
| T | T | T |

**Table for || (OR operator):** The operator || corresponds with Boolean logical operation *OR*. The operator produces a true value if either one of its two operands are true and produces a false value only when both operands are false. The following table reflects the value of || operator:

| p | q | P&&q |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

| p | q | P&&q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

- **Bitwise operator :** These operators are used to perform bitwise operations,these operations are performed on the bits of the binary pattern of the number.bitwise operators refer to testing,setting or shifting the actual bits in a byte or word, which correspond to the char and int data types.you cannot use bitwise operator on float,double,long double,void and other complex operators.

| Operator | Operation |
|---|---|
| & | And |
| \| | OR |
| ~ | NOT |
| ^ | XOR |
| >> | shift right |
| << | shift left |

**Table for ^ ( Exclusive OR):**

| p | q | P&& q |
|---|---|---|
| 0<br>1<br>1<br>0 | 0<br>0<br>1<br>1 | 0<br>1<br>0<br>1 |

**Conditional Operator**

The conditional operator evaluates an expression returning a value if that expression is true and a different value if the expression is evaluated as false. The syntax is:

```
condition ? value1 : value2
```

**For example:** In

```
7>5 ? x : y
```

Since 7 is greater than 5, true is returned and hence the value x is returned.

**Comma Operator**

*This is denoted by, and it is used to separate two or more expressions.**For example:***

```
exfor = (x=5, x+3);
```

**Here value of 5 is assigned to x and then the value of x+3 is assigned to the variable exfor. Hence, value of the variable exfor is 8.**

### sizeof() Operator

This operator accepts a single parameter and this can be a variable or data type. This operator returns the size in bytes of the variable or data type.

**For example:**

```
x = sizeof (char);
```

This returns the size of char in bytes. In this example, the size is 1 byte which is assigned to variable x.

## Data types: These are the basic data types :-

### int:

This int keyword is used to declare integers, whole numbers either positive or negative. Most of the compilers treat this with a size of 2 bytes. Its range is -32768 to 32767. Integer (2, 3, 4)

### Char:

This keyword is used to declare characters. The size of each character is 8 bits. i.e., 1 byte. The characters that can be used with this data type are ASCII characters. Its range is -128 to 128.character values (a, b, c, d).

### float:

This keyword float is used to declare floating point decimal numbers. The size of each float is 4 byte. its range is -3.4E to 3.4E. Float value(2.4,3.6,6.5).

### long:

This long keyword is used for declaring longer numbers. i.e., numbers of length 32 bits.

| keyword | Range (low) | (high) | Bytes of memory(size) |
|---|---|---|---|
| Char | -128 | 127 | 1 |
| int | -32768 | 32767 | 2 |
| long | -2147483648 | 2147483647 | 4 |
| float | 3.4 x 10-38 | 3.4 x 1038 | 4 |
| double | 1.7 x 10-308 | 1.7 x 10308 | 8 |

## Control structure:

**Conditional**

- If statement
- if –else statement
- nested if

**Looping**

- for loop
- while loop
- do while loop

**Breaking**

- break
- continue
- exit
- goto

**If statement:-** The *if* statement provides a selection control structure to execute a section of code if and only if an explicit run-time condition is met. The condition is an expression which evaluates to a boolean value, that is, either true or false.

### Syntax

```
if
( <expression> )

{

 Statement

}
```

### Semantics

- The if statement provides selection control.
- The expression is evaluated first.
- If the expression evaluates to true, the statement part of the if statement is executed.
- If the expression evaluates to false, execution continues with the next statement after the if statement.
- A boolean false, an arithmetic 0, or a null pointer are all interpreted as false.
- A boolean true, an arithmetic expression not equal to 0, or a non-null pointer are all interpreted as true.

Example:

```
#include<iostream.h>
 #include<conio.h>
Void main()
{
Int a ;
Cout<<”enter the no”;
Cin>>a;
If(n%2==0)
Cout<<”it is even no.”;
getch();
}
```

**If –else**: In this statement ,if the expression evaluated to true,the statement or the body of if statement is executed,otherwise the body of if statement is skipped and the body of else statement is executed.

```
if (condition)
{
      statement1;
else
{
      statement2; }
```

**Example:**

```
# include<iostream.h>
```

```
#include<conio.h>
Void main()
{
Clrscr();
Int  n;
Cout<<”enter the no”;
Cin>>n;
If (n%2==0)
Cout<<”it is even no”;
else
Cout<<”it is odd no”;
getch();
}
```

**Switch -** it provide multiple branch selection statement .if –else provide only two choices for selection and switch statement provide multiple choice for selection.

Syntax-

```
switch(expression)
{
Case :exp 1
First case body;
break;
Case :exp2
Second case of body:
break;
Default:
Default case body;
}
```

**Example:-**

```
#include<iosttream.h>
#include<conio.h>
 int a ;
Cout<<”enter the no”;
Cin>>a;
Switch(a)
{
Case1:cout<<”Sunday\n”;
Break;
Case1:cout<<”Sunday\n”;
break;
Case2:cout<<”monday\n”;
break;
```

```
Case3:cout<<"tuesday\n";
break;
Case4:cout<<"wednesday\n";
break;
Case5:cout<<"thrusday\n";
break;
Case6:cout<<"friday\n";
break;
Case7:cout<<"Satday\n";
break;
Default:
      Cout<<"wrong option";
}
getch();
}
```

**For loop-** In this, first the expression or the variable is initialized and then condition is checked. if the condition is false ,the loop terminates

Syntax-

for (initillization;condition;increment)

**EXAMPLE** program to print the no from 1 to 100.

```
#include<iostream.h>
# include<conio.h>
Void main ()
{
int i;
for(i=1;i<=100;i++)
Cout<<i<<"\n";
getch ();
}
```

**While loop:** This loop is an entry controlled loop and is used when the number of iteration to be performed are known in advance. The statement in the loop is executed if the test condition is true and execution continues as long as it remains true.

**Syntax-**

```
initialization;
While (condition)
{
Statement;
increment;
```

```
}
```

**Example** writes a program to print a table?

```
#include<iostream.h>
#include<conio.h>
Void main ()
{
Int  n ,i;
Cout<<"enter the no whose table is to be printed ";
Cin>>n;
i=1;
While (i<=10)
{
Cout<<n<<"x"<<i<<"="<<nxi<<"\n";
i++;
}
getch();
}
```

**Do-while -** It is bottom controlled loop. This that a do-while loop always execute at least once.

**Syntax-**

```
initillization
Do
{
Statement ;
Increement;
}
While(condition);
```

**Example**: Write a program to print the table ?

```
# include<iostream.h>
#include<conio.h>
Void main()
{
int n;
Cout<<"enter the no. whose table is to be printed";
Cin>>n;
I=1;
do
{
```

```
Cout<<n<<"x"<<i<<"="<<nxi<<"\n";
getch();
}
```

**Break statement-** The term break means breaking out of a block of code. The break statement has two use,you can use it to terminate a case in the switch statement, and you can also use it to force immediate termunation of loop,bypassing the normal loop condition test.

**Example:-**

```
#include<iosttream.h>
 #include<conio.h>
 int a ;
Cout<<"enter the no";
Cin>>a;
Switch(a)
{
Case1:cout<<"Sunday\n";
Break;
Case1:cout<<"Sunday\n";
break;
Case2:cout<<"monday\n";
break;
Case3:cout<<"tuesday\n";
break;
Case4:cout<<"wednesday\n";
break;
Case5:cout<<"thrusday\n";
break;
Case6:cout<<"friday\n";
break;
Case7:cout<<"Satday\n";
break;
Default:
     Cout<<"wrong option";
}
getch();
}
```

**Exit statement-** Exit is a function defined in the stdlib library means (stdlib.h).

The purpose of exit is to terminate the current program with a specific exit code. Its prototype is:

```
exit (exitcode);
```

The exitcode is used by some operating systems and may be used by calling programs. By convention, an exit code of 0 means that the program finished normally and any other value means that some error or unexpected results happened

**Example**

```
#include<iuostream.h>
#include<conio.h>
#include<stdlib.h>
Void main ()
{
Int n;
Cout<<"enter the no";
Cin>>n;
If (n%2==0)
{
Cout<<"it is even no";
else
Cout<<"it is odd no";
exit(O);
}
getch();
}
```

**Continue:-** The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration. For example, we are going to skip the number 5 in our countdown:

**Example**

```
#include <iostream>
 #include<conio.h>
Void main ()
{
  for (int n=10; n>0; n--)
{
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  getch();
}
```

Output- 10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!

**Goto statement:** allows to make an absolute jump to another point in the program. You should use this feature with caution since its execution causes an unconditional jump ignoring any type of nesting limitations.
The destination point is identified by a label, which is then used as an argument for the goto statement. A label is made of a valid identifier followed by a colon (:).

Generally speaking, this instruction has no concrete use in structured or object oriented programming aside from those that low-level programming fans may find for it. For example, here is our countdown loop using `goto`:

```
// goto loop example

#include <iostream>
using namespace std;

int main ()
{
  int n=10;
  loop:
  cout << n << ", ";
  n--;
  if (n>0) goto loop;
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

**What is array and also explain the different types of arrays?**

**Array :-** It is a collection of similar type of data which may be int type, char type, float type or user-defined type such as structure or class. The significance of an

array is that each array element is stored in consecutive memory locations and the array elements are accessed by their index value, which is also called subscript value.

General format of array:

**data type array name[size];**

**Single dimensional array-** In this type of array only one sub-script(index) is used in the program.

Syntax- **data type array name [size];**

**Multidimensional array-** In this type of array more than two subscript is used in the program. it is also known as array of array.

Syntax- **data type array name [row][column];**

Program- **single dimensional array**

**Write a program to print the value?**

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

```
Void main()
{
Clrscr();
Int a[10],i;
for(i=1;i<=10;i++)
{
Cout<<”enter the no:”;
Cin>>a[i];
}
for(i=1;i<=10;i++)
{
Cout<<a[i]<<endl;
}
getch();
}
```

**write a program which calculate how much spent in total ?**

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int choc [4];
cout<<”enter the no of chocolates”;
for (int i= 0;i<4;i++)
cin>>choc[i];
int total =0;
for (i=0;i<4;i++)
total =total +(choc[i] x 5);
cout<<”total cost of chochalate is :”<<total ;
getch();
}
```

**<u>Write a program two add the two matrix?</u>**

```
#include<iostream.h>
#include<iostream.h>
void main()
{
int A[3][4],B[3][4],C[3][4];
int r,c;
// read  value in  matrices
Cout<<”enter the first matrix row wise \n”;
for (r=0;r<3;r++)
```

```
{
for(c=0;c<4;c++)
{
Cin>>A[r][c];
}
}
Cout<<”enter the second matrix row wise\n”;
for (r=0;r<3;r++)
{
for (c=0;c<4;c++)
{
Cin>>B[r][c];
}
}
//addition of two matrix

for (r=0;r<3;r++)
{
for (c=0;c<4;c++)
{
C[r][c]= A[r][c]+B[r][c];
}
}
//display the matrix

for (r=0;r<3;r++)
{
for (c=0;c<4;c++)
{
Cout<<C[r][c]<<”\t”;
}
Cout<<”\n”;
}
}
getch();
}
```

**What is Function?**

A complex program contains a large list of instructions which is not easy to manage, therefore such programs are generally decompose into different

modules containing small sets of instruction that perform specific task. These modules are called function.

There are two types of function:

- Library function
- User define function

**Library function**-The function that already defines or predefines in the language is known as library function.

**User defines function**- The functions which are designed by user on the basis of requirement of a programmer are known as user defines function.

In c++ three terms are always associated with the function are:

- Function Prototype(declaration)( use semicolon;)
- Function calling (use semicolon;)
- Function definition

**Syntax of function declaration**: type function name (type parameter name);

**Simple program of addition with function:**

```
 # include<iostream>
#include<conio.h>
  Void main()
  {
Clrscr();
Int a,b,c;
```

```
Int  add (int a,int b); // function decleration
Cout<<"enter the no";
Cin>>a>>b;
C=add(a,b);  // function calling
Cout<<c;
getch();
}
Int  add (int x,int y) // function definition
{
Int  z;
Z=x+y;
return z;
}
```

**What are the different types of parameters?**

There are two types of parameters associated with functions. they are:

**(a) Actual parameter:** The parameters associated with function call are called
actual parameters.

**(b) Formal parameter:** The parameters associated with the function definition
are called formal parameters.

**Explain the different types of parameter passing.**

There are three types of parameter passing in C++. They are:

**(a) Call by value:** In this method, the actual parameters are copied into the Formal parameters and the change in the formal parameters do not affect the actual parameter. In this passing only the value so that copy of the value is sent to function,original value will not change.

**(b) Call by reference:** In this mode of parameter passing instead of passing the

Value to a function, a reference or an alias to the actual parameter is passed. The changes made in the formal parameters are reflected in the actual parameters.In this ,address of the value is passed ,so the original value will be change .

**Program call by value:- swapping program (passing the value)**

```
#include<iostream.h>
# include<conio.h>
Void main( )
{
Int x,y;
Void swap (int a,int b);
Cout<<"enter the value ";
Cin>>x>>y;
Cout<<"\n the original value of x and y"<<x<<"and" <<y;
Swap(x,y);
Cout<<"\n after swap value of x and y:'<<x<<"and"<<y;
getch( ) ;
}
Void swap (int a , int b)
{
Int c;
c=b;
b=a;
a=c;
cout<<"\n swapped x and y:"<<a<<b;
}
```

**Call by reference: swapping program(passing the address)**

```
#include<iostream.h>
# include<conio.h>
Void main( )
{
Int x,y;
Void swap (int  &a,int &b);
Cout<<"enter the value ";
Cin>>x>>y;
```

```
Cout<<"\n the original value of x and y"<<x<<"and" <<y;
Swap(x,y);
Cout<<"\n after swap value of x and y:'<<x<<"and"<<y;
getch( ) ;
}
Void swap (int &a , int &b)
{
Int c;
c=b;
b=a;
a=c;
cout<<"\n swapped x and y:"<<a<<b;
}
```

**What is Inline function?**

Inline functions are functions where the call is made to inline functions. The actual code then gets placed in the calling program.

**Reason for the need of Inline Function:**

Normally, a function call transfers the control from the calling program to the function and after the execution of the program returns the control back to the calling program after the function call. These concepts of function saved program space and memory space are used because the function is stored only in one place and is only executed when it is called. This concept of function execution may be time consuming since the registers and other processes must be saved before the function gets called. The extra time needed and the process of saving is valid for larger functions. If the function is short, the programmer may wish to place the code of the function in the calling program in order for it to be executed. This type of function is best handled by the inline function. In this situation, the programmer may be wondering "why not write the short code repeatedly inside the program wherever needed instead of going for inline function?" Although this could accomplish the task, the problem lies in the loss of clarity of the program. If the programmer repeats the same code many times, there will be a loss of clarity in the program. The alternative approach is to allow inline functions to achieve the same purpose, with the concept of functions.

**The general format of inline function is as follows:**

Inline data type function name (arguments)

The keyword inline specified in the above example, designates the function as inline function.

**Program of addition of two values with inline function.**

```
# include<iostream>
#include<conio.h>
Inline Int  add (int a,int b); // function decleration with keyword
```

**inline.**

```
   Void main()
  {
Clrscr();
Int a,b,c;
Cout<<"enter the no";
Cin>>a>>b;
C=add(a,b);  // function calling
Cout<<c;
getch();
}
Int  add (int x,int y) // function definition
{
Int  z;
Z=x+y;
return z;
}
```

**What is a friend function?**

Friend function is a special function which can access the private and protected Members of a class through the object of the same class. Friend functions are not the member functions of a class and they can be declared under any access specify.

**Program  of friend function -**

```
#include<iostream.h>
#include<conio.h>
Class car
{
Private :
Int speed ;
Char color[20];
Public:
Void input( )
{
Cout<<"enter the color";
Cin>>color;
Cout<<"enter the speed";
```

```
Cin>>speed;
}
Friend void display (car);
};
Void display(car x)
{
Cout<<"\n the color of the car is :"<<x.color;
Cout<<"\nthe speed of car is"<<x.speed;
}
Void main( )
{
Car c;
c.input( );
display(c);
garch ( );
}
```

**What is function overloading?**

C++ enables several functions of the same name to be defined, as long as these functions have different sets of parameters (at least as far as their types are concerned). This capability is called function overloading. When an overloaded function is called, the C++ compiler selects the proper function by examining the number, types and order of the arguments in the call. **Function overloading is commonly used to create several functions of the same name that perform similar tasks but on different data types or arguments.**

**Program to find the volume of cube, volume of cylinder, volume of rectangular box, using function overloading?**

```
# include<iostream.h>
#include<conio.h>
Int  volume(int);
float volume(float , int);
long int volume(long int,int,int);
void main( )
{
```

```
Cout<<"volume of cube"<<volume (10);
Cout<<"volume of cylinder"<<volume (4.5, 5);
Cout<<"volume of Rectangular box"<<volume (8, 7, 3);
getch ( );
}
Int volume(int a)
{
return(a*a*a);
}
float volume(float r,int h)
{
return (3.14 * r * r * h);
}
long int volume(long int  l, int  b, int h);
{
return(l*b*h);
}
```

**Operator overloading**: Allows existing C++ operators to be redefined so that they work on objects of user-defined classes. Overloaded operators are syntactic sugar for equivalent function calls. They form a pleasant facade that doesn't add anything fundamental to the language (but they can improve understandability reduce maintenance costs).

**Virtual function**

Virtual functions are functions with the same function prototype that are defined throughout a class hierarchy. At least the base class occurrence of the function is preceded by the keyword virtual. Virtual functions are used to enable generic processing of an entire class hierarchy of objects through a base class pointer. For example, in a shape hierarchy, all shapes can be drawn. If all shapes are derived from a base class Shape which contains a virtual draw function, then generic processing of the hierarchy can be performed by calling every shape's draw generically through a base class Shape pointer.

**What is a return statement in function and how many return statements can be used In a function?**

**Ans:** Return statement is used to return the value from the function definition to the function call in the program. Only one return statement is executed in the function.

**Class:** it is an important concept of object oriented programming. It is a user defines data type which contains data member & member function.
It is collection of various kind of object. It is define by class keyword. It also an Important feature of object oriented programming language. For ex-fruit is a class And apple, mango, banana are its object.

It contains data members and member function which are declared under class.

There are three types of data members declare in class-

**Public:** In this data members and member function are accessible outside the class.

**Private:** Data members and member function are not accessible outside the class.

**Protected:** Data members and member function are only available to derived class.

**Program to add two values using class?**

```
#include<iostream.h>
#include<conio.h>
Class add
{
Public:
int a,b,c;
Void input ();
Void output ();
};

Void adds:: input ()
{
Cout<<"enter the no";
Cin>>a>>b;
}
```

```
Void add:: output ()
{
C=a+b;
Cout<<c;
}
Void main ()
{
Clrscr ();
Add d;
d.input ();
d.output ();
getch ();
}
```

**What are the differences between a structure and a class?**

Ans: There is no difference in structure and a class except that class members are **private** by default and structure members are **public** by default. And in class a keyword **class** is used and in structure **struct** keyword is used

**Structure program:**
**To print the student name, branch, roll no and age?**

```
# include<iostream.h>
#include<conio.h>
Struct student
{
Char name[10];
Char branch[10];
Int rollno;
Int age;
};
Void main( )
{
Student s;
Cout<<"\n enter the name of student";
Cin>>s.name;
Cout<<"\n enter the branch of student";
Cin>>s.branch;
Cout<<"\n enter the age";
Cin>>s.age;
getch();
}
```

**Inheritance:** Inheritance is the process by which new classes called derived class are created from existing class called base class. The derived class has all the features of base class and the programmer can choose to add new features specific to the newly created derived class.

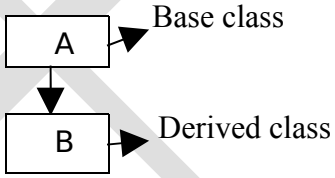
For example: a programmer can creat a base class “fruit” and derived class as “mango”, “apple”, “banana”,”orange”.

In this,each of these derived class has all the features of base class (fruit) with additional attributes or specific to these newly created derived class.in this way mango would have its own features and apple would have their own.

## Types of inheritance:

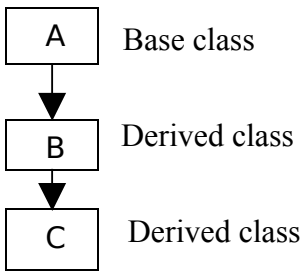
**(a) Single-Level Inheritance**

When one class is derived only from one base class then such inheritance is called single-level inheritance. The single-level inheritance is shown below.



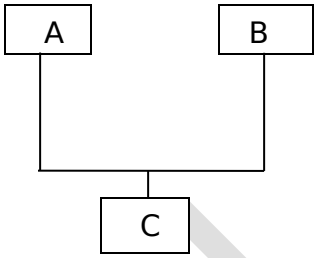

**(b) Multilevel Inheritance**

When the single-level inheritance is extended to more levels then it is called multilevel inheritance. In this inheritance one class is derived from another derived class and the level of derivation can be extended to any number of levels. For example, class C is derived from another derived class B which itself is derived from class A.



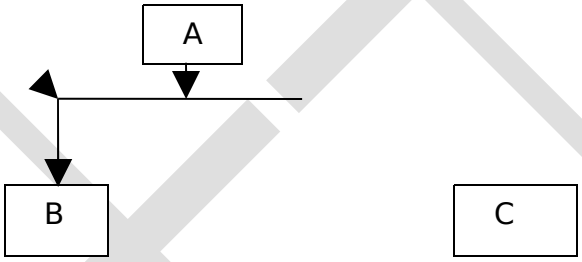

**(c) Multiple Inheritance**

When single class inherits the properties from more than one base class, it is called the multiple inheritance. In other words we can say that multiple inheritance means that one class can have more than one base class. It allows us to combine features of several existing classes into a single class as shown below




**(d) Hierarchical Inheritance**

When many subclasses inherit properties from a single base class, it is called as hierarchical inheritance. The base class contains the features that are common to the subclass and a subclass can inherit all or some of the features from the base class as shown below




**(e) Hybrid Inheritance**

It is a combination of multiple inheritances and the hybrid inheritance. In hybrid Inheritance a class inherits from a multiple base class which itself inherits from a single base class. This form of inheritance is known as hybrid inheritance. It is shown below