Q.1] Write an algorithm, draw a flowchart and develop a C++ program to the demonstrate the use of single inheritance.

#include<iostream.h>

#include<conio.h>

const int MAX=100;

class basicinfo

{

private:

char name[20];

long int rno;

public:

void getdata();

void display();

};

class physicalfit:public basicinfo

{

private:

float height;

float weight;

public:

void getdata();

void display();

};

void basicinfo::getdata()

{

cout<<"\n enter a name:";

cin>>name;

cout<<"\n enter roll number:";

cin>>rno;

}

void basicinfo::display()

{

cout<<name<<"\t";

cout<<rno<<"\t";

}

void physicalfit::getdata()

{

basicinfo::getdata();

cout<<"height";

cin>>height;

cout<<"weight";

cin>>weight;

}

void physicalfit::display()

{

basicinfo::display();;

cout<<height<<"\t";

cout<<weight<<"\t";

}

void main()

{

physicalfit a[MAX];

int l,n,i;

clrscr();

cout<<"\n enter the number of student:";

cin>>n;

cout<<"\n enter the information:";

for(i=0;i<=n-1;i++)

{

cout<<"\n record:"<<i+1;

a[i].getdata();

}

cout<<endl;

cout<<"name rollnumber height weight\n";

for(i=0;i<=n-1;i++)

{

a[i].display();

cout<<"\n";

}

cout<<endl;

getch();

}

Output:-

enter the number of student:2

enter the information:

record:1

enter a name: Aashish

enter roll number:25

height170

weight57

record:2

enter a name:Ankita

enter roll number:26

height160

weight53

name rollnumber height weight

Aashish 25 170 57

Ankita 26 160 53

**Friend Class** A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

**Function** Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:   
a) A member of another class   
b) A global function

Following are some important points about friend functions and classes:   
**1)** Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.  
**2)** Friendship is not mutual. If class A is a friend of B, then B doesn’t become a friend of A automatically.  
**3)** Friendship is not inherited

Q.2] Write an algorithm, draw a flowchart and develop a C++ program to find largest amoung two value using friend function.

#include<iostream.h>

#include<conio.h>

class biggest

{

private:

int a,b,c,large;

public:

void getdata();

friend int big(biggest abc); /\* friend function\*/

};

void biggest::getdata()

{

cout<<"enter any 3 number:";

cin>>a>>b>>c;/\*5,8,4\*/

}

int big(biggest abc)

{

abc.large=abc.a;/\*5\*/

if(abc.b>abc.large)

{

abc.large=abc.b;/\*8\*/

}

if(abc.c>abc.large)/\*4\*/

{

abc.large=abc.c;

}

cout<<"biggest number is="<<abc.large;

return(0);

}

void main()

{

class biggest obj;

clrscr();

obj.getdata();

big(obj);

getch();

}

Output:-

enter any 3 number:2

5

7

biggest number is=7

Q.3] Write an algorithm, draw a flowchart and develop a C++ program in which a function is passed address of two variables and then alter its contents.

#include<iostream.h>

#include<conio.h>

int main()

{

clrscr();

float add(float,float);

float sub(float,float);

float action(float(\*)(float,float),float,float);

float (\*ptrf)(float,float);

float a,b,value;

char ch;

cout<<"passing a function to another function:\n";

cout<<"\n enter any two number\n";

cin>>a>>b;

cout<<"a addition"<<endl;

cout<<"s substraction"<<endl;

cout<<"option.please?\n";

cin>>ch;

if(ch=='a')

ptrf=&add;

else

ptrf=&sub;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

value=action(ptrf,a,b);

cout<<"answer="<<value<<endl;

getch();

float add(float x,float y)

{

float ans;

ans=x+y;

return(ans);

}

float sub(float x,float y)

{

float ans;

ans=x-y;

return(ans);

}

float action(float(\*ptrf)(float,float),float x,float y)

{

float answer;

answer=(\*ptrf)(x,y);

return(answer);

}

Output:-

passing a function to another function:

enter any two number

24

25

a addition

s substraction

option.please?

a

a=24, b=25

answer=49

# C++ Recursion

When function is called within the same function, it is known as recursion in C++. The function which calls the same function, is known as recursive function.

A function that calls itself, and doesn't perform any task after function call, is known as tail recursion. In tail recursion, we generally call the same function with return statement.

Q.4] Write an algorithm, draw a flowchart and develop a C++ program to display Fibonacci series (i) using recursion,(ii) using iteration.

#include<iostream.h>

#include<conio.h>

int main()

{

int sum(int);/\* function declaration\*/

int n,temp;

clrscr();

cout<<"enter any integer number\n";

cin>>n; //4

temp=sum(n);// function calling

cout<<"1+2+3...."<<n;

cout<<"and its sum="<<temp<<"\n";

getch();

}

int sum(int n)

{

int sum(int);

int value=0;

if(n==0)//4

return(value);

else;

value=n+sum(n-1);/\* 4+3

return(value);7

}

**Output:-** enter any integer number 12

12+11+10+9+8+7+6+5+4+3+2+1=

# Factorial program in C++

**Factorial Program in C++:** Factorial of n is the product of all positive descending integers. Factorial of n is denoted by n!. For example:

1. 4! = 4\*3\*2\*1 = 24
2. 6! = 6\*5\*4\*3\*2\*1 = 720

Q.5] Write an algorithm, draw a flowchart and develop a C++ program to calculate factorial of a number.

#include<iostream.h>

#include<conio.h>

void main()

{

long int fact(long int);// function declartion

long int x,n;

clrscr();

cout<<"enter any number:";

cin>>n;

x=fact(n);// function calling

cout<<"values="<<n<<"\n and its factorial="<<x;

cout<<"\n";

getch();

}

long int fact(long int n)

{

long int fact(long int);

int value=1;

if(n==1)

{

value=n\*fact(n-1);

return(value);

}

}

Output:-

enter any number:65

values=65

and its factorial=1138

Every object in C++ has access to its own address through an important pointer called **this** pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a **this** pointer, because friends are not members of a class. Only member functions have a **this** pointer.

The Class Constructor

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Q.6] Write an algorithm, draw a flowchart and develop a C++ program to demonstrate the use of this pointer.

#include <iostream>

using namespace std;

class Box {

public:

// Constructor definition

Box(double l = 2.0, double b = 2.0, double h = 2.0) {

cout <<"Constructor called." << endl;

length = l;

breadth = b;

height = h;

}

double Volume() {

return length \* breadth \* height;

}

int compare(Box box) {

return this->Volume() > box.Volume();

}

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

int main(void) {

Box Box1(3.3, 1.2, 1.5); // Declare box1

Box Box2(8.5, 6.0, 2.0); // Declare box2

if(Box1.compare(Box2)) {

cout << "Box2 is smaller than Box1" <<endl;

} else {

cout << "Box2 is equal to or larger than Box1" <<endl;

}

return 0;

}

Output:

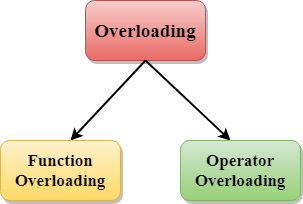
Constructor called.

Constructor called.

Box2 is equal to or larger than Box1

# C++ Overloading (Function and Operator)

If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading.



## C++ Operators Overloading

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

Syntax of Operator Overloading

1. return\_type class\_name  : : operator op(argument\_list)
2. {
3. // body of the function.
4. }

Where the **return type** is the type of value returned by the function.

**class\_name** is the name of the class.

**operator op** is an operator function where op is the operator being overloaded, and the operator is the keyword

Q.7] Write an algorithm, draw a flowchart and develop a C++ program to perform unary operator.

Overloading.

#include<iostream.h>

#include<conio.h>

struct fibonacci

{

public:

unsigned long int f0,f1,fib;

fibonacci();

void operator++();

void display();

};

fibonacci::fibonacci()

{

f0=1;

f1=1;

fib=f0=f1;// 1+1=2

}

void fibonacci::display()

{

cout<<"\t"<<fib;

}

void fibonacci::operator++()

{

f0=f1;// 1

f1=fib;//2

fib=f0+f1;// 1+2=3

}

int main()

{

fibonacci obj;

int n;

clrscr();

cout<<"enter the term\n";

cin>>n;

cout<<obj.f0<<"\t"<<obj.f1;

for(int i=1;i<=n;++i)

{

obj.display();

++obj;

}

cout<<endl;

getch();

}

Output:-

enter the term

12

1 1 1 2 3 5 8 13 21 34

55 89 144 233

C++ Function Overloading

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences compiler can differentiate between the functions.

Q.8] Write an algorithm, draw a flowchart and develop a C++ program to create a class Triangle.Include overload functions for calculating area. Overload assignment operator and equality operator.

#include<iostream.h>

#include<conio.h>

#include<math.h>

class triangle

{

float b,h,area;

public:

float getarea()

{

area=0.5\*(b\*h);

return(area);

}

void set\_base (float base)

{

b=base;

}

void set\_height (float height)

{

h=height;

}

void operator=(triangle t)

{

b=t.b;

h=t.h;

}

void operator==(triangle t)

{

if(t.b==t.h)

{

area=(sqrt(3/4)\*pow(t.b,2));

cout<<"area of equilateral triangle:"<<t.area;

}

}

};

void main()

{

clrscr();

triangle t1,t2;

t1.set\_base(2.0);

t2.set\_height(3.0);

cout<<"\n area;"<<t1.getarea();

t2=t1;

cout<<"\n t2:area:"<<t2.getarea();

t2.operator==(t1);

getch();

}

Output:-

area:4.14167e-33

t2:area:4.14167e-33

# What is Memory Management?

Memory management is a process of managing computer memory, assigning the memory space to the programs to improve the overall system performance.

## Why is memory management required?

As we know that arrays store the homogeneous data, so most of the time, memory is allocated to the array at the declaration time. Sometimes the situation arises when the exact memory is not determined until runtime. To avoid such a situation, we declare an array with a maximum size, but some memory will be unused. To avoid the wastage of memory, we use the new operator to allocate the memory dynamically at the run time.

## Memory Management Operators

In [C language](https://www.javatpoint.com/c-programming-language-tutorial), we use the **malloc()** or **calloc()** functions to allocate the memory dynamically at run time, and free() function is used to deallocate the dynamically allocated memory. [C++](https://www.javatpoint.com/cpp-tutorial) also supports these functions, but C++ also defines unary operators such as **new** and **delete** to perform the same tasks, i.e., allocating and freeing the memory.

### New operator

A **new** operator is used to create the object while a **delete** operator is used to delete the object. When the object is created by using the new operator, then the object will exist until we explicitly use the delete operator to delete the object. Therefore, we can say that the lifetime of the object is not related to the block structure of the program.

Q.9] Write an algorithm, draw a flowchart anfd develop a C++ program to find sum of n elements .Entered by the user.To write this program, allocate memory dynamically using malloc()/calloc()Functions or new operator.

#include<iostream.h>

#include<conio.h>

void main()

{

int \*ptr\_i=new int(25);

float \*ptr\_f=new float(-10.12347);

char \*ptr\_c=new char('a');

double \*ptr\_d=new double(1234.5667);

clrscr();

cout<<"contents of the pointer"<<endl;

cout<<"intger="<<"ptr\_i"<<endl;

cout<<"floating point value="<<"ptr-1"<<endl;

cout<<"double="<< \*ptr\_d<<endl;

delete ptr\_i;

delete ptr\_f;

delete ptr\_c;

delete ptr\_d;

getch();

}

Output:-

contents of the pointer

intger=ptr\_i

floating point value=ptr-1

double=1234.5667

Q.10] Write an algorithm, draw a flowchart anfd develop a C++ program to create Marksheet .Using multilevel inheritance.

#include<iostream.h>

#include<conio.h>

class student

{

protected:

int rno;

public:

void getno(int);

void putno();

};

void student::getno(int a)

{

rno=a;

}

void student::putno()

{

cout<<"roll numbear:"<<rno<<"\n";

}

class test:public student

{

protected:

float sub1;

float sub2;

public:

void getmarks(float,float);

void putmarks(void);

};

void test::getmarks(float x,float y)

{

sub1=x;

sub2=y;

}

void test::putmarks()

{

cout<<"marks in sub1="<<sub1<<"\n";

cout<<"marks in sub2="<<sub2<<"\n";

}

class result:public test

{

float tot;

public:

void display(void);

};

void result::display(void)

{

tot=sub1+sub2;

putno();

putmarks();

cout<<"total="<<tot<<"\n";

}

void main()

{

clrscr();

result student;

student.getno(111);

student.getmarks(75,76);

student.display();

getch();

}

Output:-

roll numbear:111

marks in sub1=75

marks in sub2=76

total=151