W1)**Inheritance:**

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its base/parent object automatically. In such way, can reuse, extend or modify the attributes and behaviours which are defined in other class.

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.

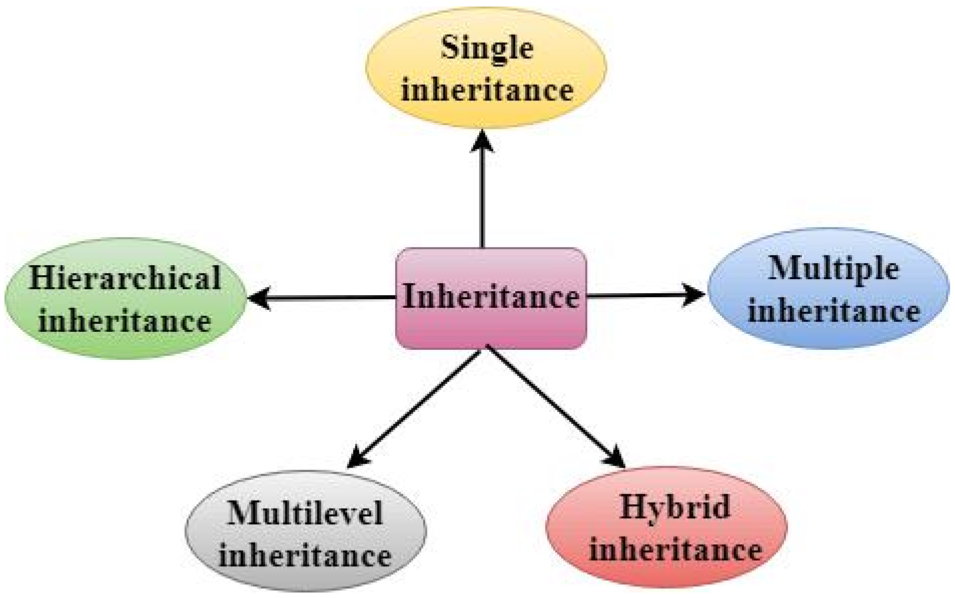
Advantage of C++ Inheritance

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.

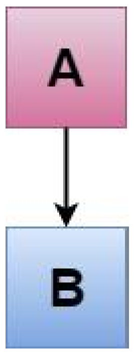
- When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class. Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.
- When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

Note:

- In C++, the default mode of visibility is private.
- The private members of the base class are never inherited.

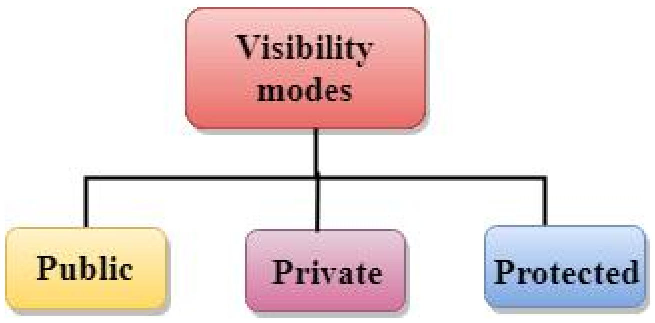
C++ Single Inheritance

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.

Visibility modes can be classified into three categories:



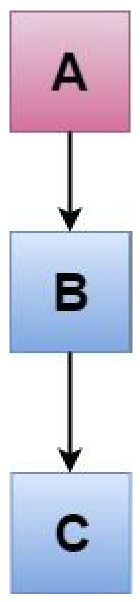
- **Public**: When the member is declared as public, it is accessible to all the functions of the program.
- **Private**: When the member is declared as private, it is accessible within the class only.
- **Protected**: When the member is declared as protected, it is accessible within its own class as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

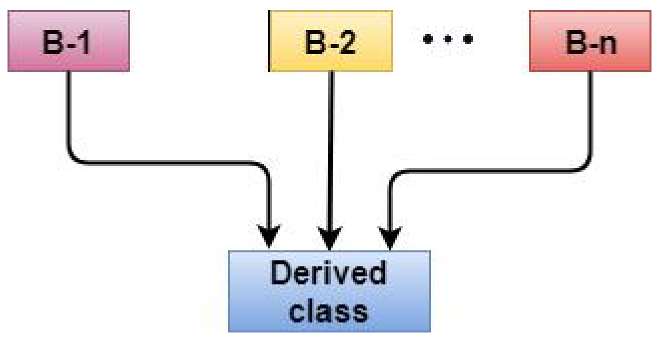
C++ Multilevel Inheritance

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



C++ Multiple Inheritance

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

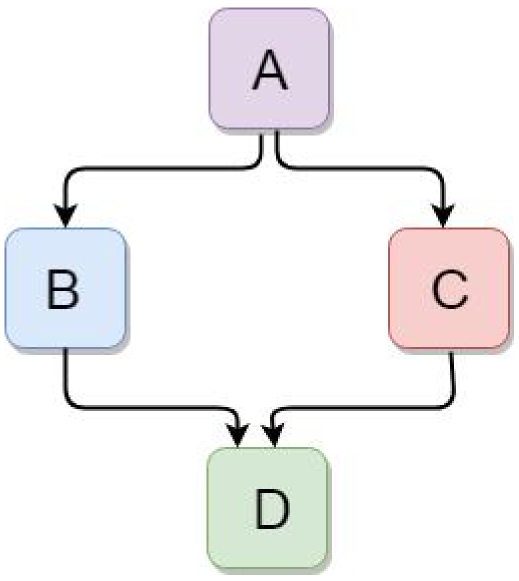


Syntax of the Derived class:

```
class D : visibility B-1, visibility B-2, ?
{
    // Body of the class;
}
```

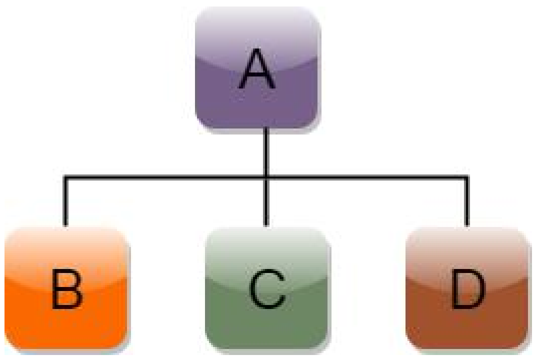
C++ Hybrid Inheritance

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



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.
}
```

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 roll number height weight
Aashish 25     170     57
Ankita  26     160     53
```

Q.2] Write an algorithm, draw a flowchart and develop a C++ program to find largest among 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);

};

void biggest::getdata()

{

cout<<"enter any 3 number:";

cin>>a>>b>>c;

}

int big(biggest abc)

{

abc.large=abc.a;

if(abc.b>abc.large)

{

abc.large=abc.b;

}

if(abc.c>abc.large)

{

abc.large=abc.c;

}

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

return(0);
```

Q.2] Write an algorithm, draw a flowchart and develop a C++ program to find largest among 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);

};

void biggest::getdata()

{

cout<<"enter any 3 number:";

cin>>a>>b>>c;

}

int big(biggest abc)

{

abc.large=abc.a;

if(abc.b>abc.large)

{

abc.large=abc.b;

}

if(abc.c>abc.large)

{

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);
     }
```

<u>Output</u>:-

```
enter any two number
24
25
a addition
s substraction
option.please?
a
a=24,   b=25
 answer=49
```

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);
int n,temp;
clrscr();
cout<<"enter any integer number\n";
cin>>n;
temp=sum(n);
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)
return(value);
else
value=n+sum(n-1);
return(value);
}
```

<u>Output:-</u>
```
enter any integer number 12
          1+2+3....12and its sum=78
```

Q.5] Write an algorithm, draw a flowchart and develop a C++ program to demonstrate the use of this pointer.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  long int fact(long int);
  long int x,n;
  clrscr();
  cout<<"enter any number:";
  cin>>n;
  x=fact(n);
  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.

```
class ClassName {



   private:

    int dataMember;



   public:

     method(int a) {



 // this pointer stores the address of object obj and access dataMember

       this->dataMember = a;

       ... .. ...

     }

}



int main() {



  ClassName obj;

  obj.method(5);

  ... .. ...
}
```

Applications of this pointer

1. Return Object

One of the important applications of using *this* pointer is to return the object it points. **For example**, the statement

```
return *this;
```

inside a member function will return the object that calls the function.

2. Method Chaining

After returning the object from a function, a very useful application would be to chain the methods for ease and a cleaner code.

For example,

```
positionObj->setX(15)->setY(15)->setZ(15);
```

Here, the methods *setX*, *setY*, *setZ* are chained to the object, *positionObj*. This is possible because each method return *this pointer.
This is equivalent to

```
positionObj->setX(15);

positionObj->setY(15);

positionObj->setZ(15);
```

3. Distinguish Data Members

Another application of *this* pointer is distinguishing data members from local variables of member functions if they have same name. **For example**,

```
#include<iostream>
```

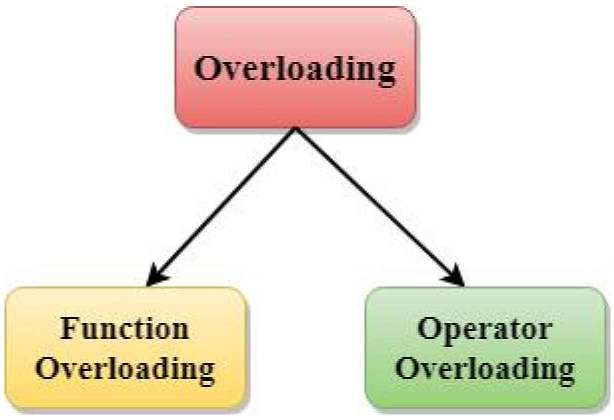
```
#include<conio.h>

using namespace std;


class student

{

   char name[100];

   int age,roll;

   float percent;

   public:

      void getdata()

      {

         cout<<"Enter data"<<endl;

         cout<<"Name:";

         cin>>name;

         cout<<"Age:";

         cin>>age;

         cout<<"Roll:";

         cin>>roll;

         cout<<"Percent:";

         cin>>percent;

         cout<<endl;

      }
```

```
        student & max(student &s1,student &s2)
        {
            if(percent>s1.percent && percent>s2.percent)
                return *this;
            else if(s1.percent>percent && s1.percent>s2.percent)
                return s1;
            else if(s2.percent>percent && s2.percent>s1.percent)
                return s2;
        }
        void display()
        {
            cout<<"Name:"<<name<<endl;
            cout<<"Age:"<<age<<endl;
            cout<<"Roll:"<<roll<<endl;
            cout<<"Percent:"<<percent;
        }
};

int main()
{
    student s,s1,s2,s3;
    s1.getdata();
    s2.getdata();

    s3.getdata();

    s=s3.max(s1,s2);

    cout<<"Student with highest percentage"<<endl;

    s.display();

    getch();

    return 0;

}
```

Operator Overloading:

Types of overloading in C++ are:

- Function overloading
- Operator overloading



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.

The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

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.

The advantage of Operators overloading is to perform different operations on the same operand.

Operator that cannot be overloaded are as follows:

- Scope operator (::)
- Sizeof
- member selector(.)

- member pointer selector(*)
- ternary operator(?:)

Syntax of Operator Overloading

```
return_type class_name  : : operator op(argument_list)
{
     // body of the function.
}
```

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.

Rules for Operator Overloading

- Existing operators can only be overloaded, but the new operators cannot be overloaded.
- The overloaded operator contains atleast one operand of the user-defined data type.
- We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
- When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
- When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

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;

}

          void fibonacci::display()

{

          cout<<"\t"<<fib;

}

          void fibonacci::operator++()

{

          f0=f1;

          f1=fib;

          fib=f0+f1;

}

          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    2    3    5    8    13    21    34    55
89    144    233    377
```

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();
}
```

<u>Output:-</u>

```
 area:4.14167e-33
 t2:area:4.14167e-33
```

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
```

Page No.
Date / /

1) Start

2) declare the base class as
   basic - info

3) define & declare getdata ()
   ~~& display~~ data to get stu infn
   & display data () to display
   infn

4) declare derived class as
   physical fit

5) define & declare get data ()
   for heigH, wt, display ()
   to display ~~[illegible]~~ ht wt

6) Read no of stu

7) call func getdat data for
   name, rno, ht, wt

8) display ~~func~~ ~~data~~
   n, rn, ht. call & 9
   display.

9) Stop.