Introduction to C++

- Inheritance
- Advantage of Inheritance
- Types of Inheritance
- Ambiguity in Multiple Inheritance
- Types of Derivation
- Order of constructor call in Single inheritance
- Order of constructor call in Multiple inheritance

Inheritance

Inheritance:

Inheritance is one of the most important concepts in object-oriented programming. Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.

- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
- The idea of inheritance implements the **IS-A** relationship.

e.g. Manager is an Employee.

Syntax of Inheritance:

```
class subclass_name : access-specifier superclass_name
{
// data members
// methods
}
```

where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Inheritance

Advantages of inheritance:

- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

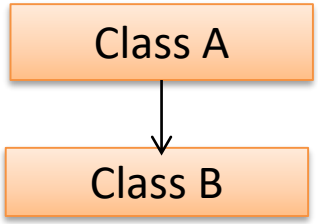
Types of Inheritance are as follows:

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

Inheritance

1. Single inheritance:

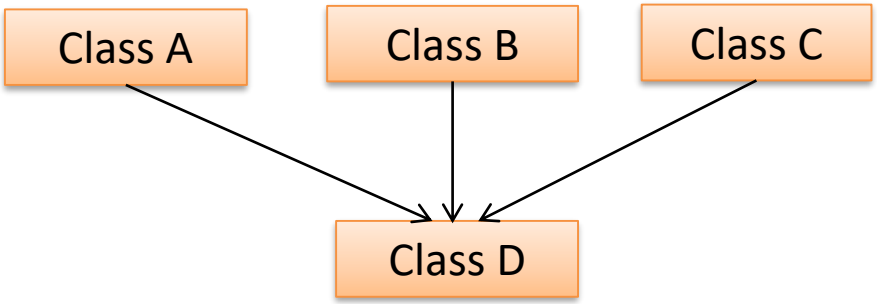
In single inheritance there exists single base class and single derived class.



As we can see in the diagram ,here class A is the super class & class B is the sub class which is acquiring the features of class A.

2. Multiple inheritance:

In this type of inheritance, one Derived class is formed by acquiring the properties of multiple Base classes.

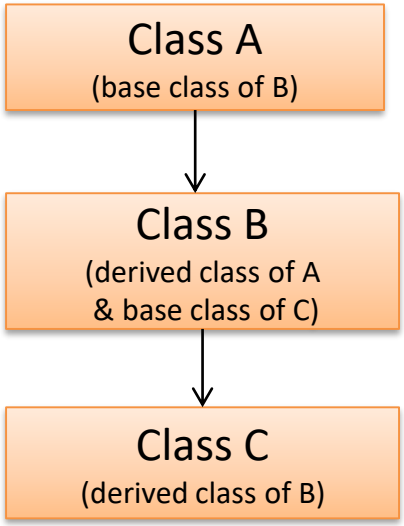


Inheritance

As shown in the diagram,class A,B & C are the base classes for class D.This means class D is acquiring the properties of A,B & C classes.

If we want to create a class with multiple base classes, we have to use following syntax:
Class DerivedClass: accessSpecifier BaseClass1, BaseClass2, …, BaseClassN

3. Multilevel inheritance:
Multilevel inheritance represents a type of inheritance when a Derived class is a base class for another class. In other words, deriving a class from a derived class is known as multi-level inheritance.

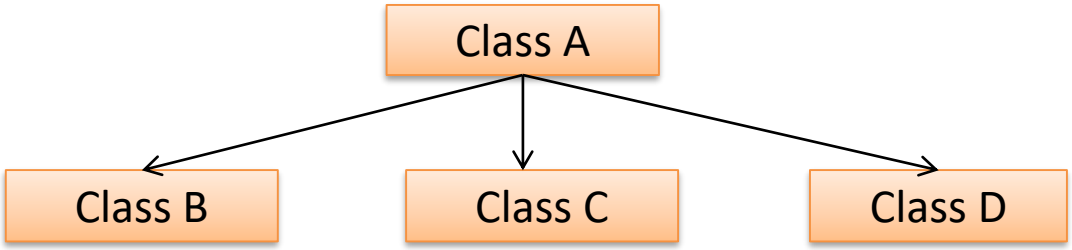


As we can see in the diagram,Class A is a parent of Class B and Class B is a parent of Class C.

Inheritance

4. Hierarchical inheritance:

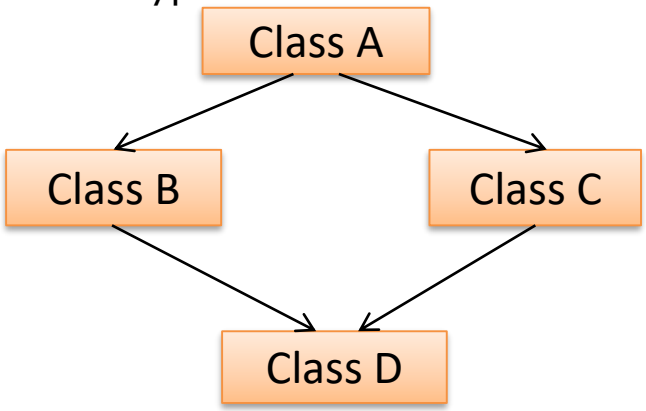
In this type of inheritance, multiple subclass are formed by acquiring the properties of one Base class.



As we can see in the diagram, Class B,C & D are acquiring the properties of same base class A.

5. Hybrid inheritance:

Combination of any inheritance type.



Inheritance

Example of Single Inheritance:

```
#include<iostream>
using namespace std;
class Student
{public:
int rno;
char name[20];
void getData()
{
cout<<"Enter RollNo :-";
cin>>rno;
cout<<"Enter Name :-";
cin>>name;
}};
class Marks : public Student
{public:
int m1,m2,m3,tot;
void getMarks()
{cout<<"Enter Marks 1 :- ";
cin>>m1;
cout<<"Enter Marks 2 :- ";
```

Inheritance

```
cin>>m2;
cout<<"Enter Marks 3 :- ";
cin>>m3;
}
void display()
{tot=m1+m2+m3;
cout<<"Roll No. is "<<rno<<endl;
cout<<"Name is "<<name<<endl;
cout<<"Marks1 is "<<m1<<endl;
cout<<"Marks2 is "<<m2<<endl;
cout<<"Marks3 is "<<m3<<endl;
cout<<"Total marks "<<tot<<endl;
}
};
int main()
{
Marks m;
m.getData();
m.getMarks();
m.display();
}
```

Inheritance

Example of Multiple Inheritance:

```
#include<iostream>
using namespace std;
class student
{
   protected:
     int rno,m1,m2;
   public:
            void get()
           {
                    cout<<"Enter the Roll no :";
                    cin>>rno;
                    cout<<"Enter the two marks   :";
                    cin>>m1>>m2;
           }
};
class sports
{
   protected:
     int sm;                 // sm = Sports mark
   public:
            void getsm()
           {
             cout<<"\nEnter the sports mark :";
             cin>>sm;
            }};
```

Inheritance

```
class statement:public student,public sports
{
   int tot,avg;
   public:
   void display()
           {
             tot=(m1+m2+sm);
             avg=tot/3;
             cout<<"\n\n\tRoll No    : "<<rno<<"\n\tTotal     : "<<tot;
            cout<<"\n\tAverage    : "<<avg;
           }
};
int main()
{
  statement obj;
  obj.get();
  obj.getsm();
  obj.display();
}
```

Inheritance

Ambiguity in multiple inheritance:

- Can arise when two base classes contain a function of the same name .
- Can arise when the derived class has multiple copies of the same base class.

Case 1:Ambiguity can arise when two base classes contain a function of the same name :

Example:

```
#include<iostream>
 using namespace std;
class base1
{       public:
        void disp()
        {
        cout << "Base1" <<endl;
        }
};
class base2
{       public:
        void disp()
        {
        cout << "Base2"<<endl;
        }
};
```

Inheritance

```
class derived : public base1,public base2
{
        // ….
};
int main()
{
        derived Dvar;
        Dvar.disp();    //Ambiguous function call
        return 0;
}
```

This can be resolved in two ways:

1. By using the scope resolution operator:

As we can see in the above eg.,compiler does not know which disp() function to invoke,that of the base1 class or base2 class.This ambiguity is resolved by using scope resolution operator as shown below:

```
Dvar.base1::disp();      //invokes disp() of base1
Dvar.base2::disp();      //invokes disp() of base2
```

Inheritance

2. By overriding the function in the derived class:

Example:

```
#include<iostream>
 using namespace std;
 class base1
 {          public:
            void disp()
            {
            cout << "Base1" <<endl;
            }
 };
 class base2
 {          public:
            void disp()
            {
            cout << "Base2"<<endl;
            }
 };
```

Inheritance

```
class derived : public base1,public base2
{
        public:
        void disp()
        {
                base1::disp();
                base2::disp();
                cout<<"Derived class";
        }
};
int main()
{
        derived Dvar;
        Dvar.disp();    //No Ambiguity
}
```

Output:

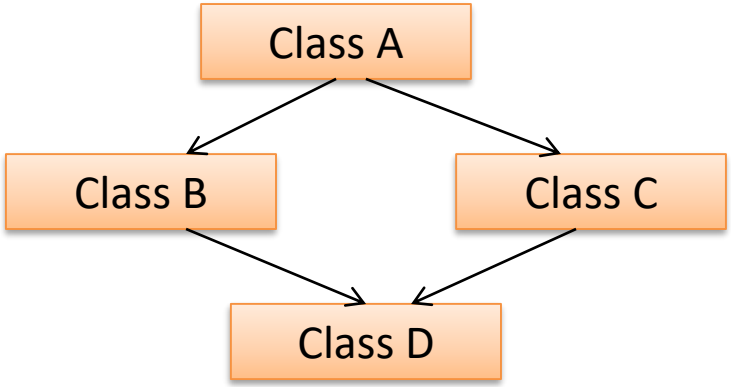
Base1

Base2

Derived class

Inheritance

Case 2: Ambiguity can arise when the derived class has multiple copies of the same base class:



As we can see in the diagram,Class D is inheriting the members of class A through class B and through class C.

Inheritance

Example:

```
#include<iostream>
using namespace std;
class A
{ public:
            int Avar;
};
class B:public A
{ public:
            int Bvar;
};
class C:public A
{
  public:
            int Cvar;
};
class D:public B,public C
{
  public:
            int Dvar;
};
```

Inheritance

```
int main()
{
  D obj;
  obj.Avar=10;    //This is ambiguous
}
```

This can be resolved in two ways:

1. By using the scope resolution operator:

As we can see in the program,ambiguity is caused because class D inherits one copy of Avar through class B & another through C.This ambiguity can be resolved by using scope resolution operator as shown below:

```
int main()
{
D obj;
obj.B::Avar=10;    //refers to Avar inherited from class B
obj.C::Avar=20;    //refers to Avar inherited from class C
}
```

Inheritance

2. By using a virtual base class:

The idea behind virtual base class is to have only one copy of the base class members in memory.Inheriting a class more than once through multiple paths creates multiple copies of the base class members.Thus by declaring base class inheritance as virtual,only one copy of the base class is inherited.

Example:

```
#include<iostream>
using namespace std;
class A
{ public:
            int Avar;
};
class B:virtual public A
{ public:
            int Bvar;
};
class C:virtual public A
{
  public:
            int Cvar;
};
```

Inheritance

```
class D:public B,C
{
  public:
            int Dvar;
};
int main()
{
  D obj;
  obj.Avar=10;    //No ambiguity
}
```

Inheritance

Types of Derivation:
When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier.
We hardly use protected or private inheritance, but public inheritance is commonly used.

While using different type of inheritance, following rules are applied:

public Inheritance: When deriving a class from a public base class:
public members of the base class become public members of the derived class.
protected members of the base class become protected members of the derived class.
A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
Syntax of public derivation is as follows:

```
class base_class
{
…
}
class derived_class:public base_class
{
…
}
```

Inheritance

protected Inheritance: When deriving a class from a protected base class:
public and protected members of the base class become protected members of the derived class.
Syntax of protected derivation is as follows:

```
class base_class
{
…
}
class derived_class:protected base_class
{
…
}
```

private Inheritance: When deriving a class from a private base class:
public and protected members of the base class become private members of the derived class.
Syntax of private derivation is as follows:

```
class base_class
{
…
}
class derived_class:private base_class
{
…
}
```

Inheritance

Order of constructor call:
When a default or parameterized constructor of a derived class is called, the default constructor of a base class is called automatically. As we create an object of a derived class, first the default constructor of a base class is called after that constructor of a derived class is called.

Points to remember:

1. Whether derived class's default constructor is called or parameterized is called, base class's default constructor is always called inside them.
2. To call base class's parameterized constructor inside derived class's parameterized constructor, we must mention it explicitly while declaring derived class's parameterized constructor.

Inhritance

Example: Order of constructor call in case of default constructor:

```
#include<iostream>
using namespace std;
class Base
{
  int x;
  public:
  Base()
  {
            cout << "Base default constructor"<<endl;
  }
};
class Derived : public Base
{
  int y;
  public:
  Derived()
  {
    cout << "Derived default constructor"<<endl;
  }
};
```

Inheritance

```
int main()
{
  Derived d1;
}
```

Output:

Base default constructor

Derived default constructor

Inheritance

To call parameterized constructor of a base class we need to call it:

Example:

```
#include<iostream>
using namespace std;
class Person
{
        string lastName;
        int yearOfBirth;
        public:
        Person()
        {
                cout << "Default constructor of base class called" << endl;
        }
        Person(string lName, int year)
        {
                cout << "Parameterized constructor of base class called" << endl;
                lastName = lName;
                yearOfBirth = year;
        }
};
class Student :public Person
{       string university;
        public:
        Student()
        {
                cout << "Default constructor of Derived class called" << endl;
        }
```

Inheritance

```
Student(string lName, int year, string univer):Person(lName,year)

{
        cout << "Parameterized constructor of Derived class called" << endl;
        university = univer;
}
};
int main()
{
        Student s("kumar",1990,"XYZ");
}
```

Output:

Default constructor of base class called

Parameterized constructor of base class called

Parameterized constructor of Derived class called

Inheritance

In case of Multiple inheritance,constructor is called in the following order:

- virtual base class constructors-in the order of inheritance.
- Non-virtual base class constructors-in the order of inheritance.
- Member object's constructor-in the order of declaration.
- Derived class constructor.

Destructors are invoked in the reverse order of the invocation of constructors.

Inheritance

Example:

```
#include<iostream>
using namespace std;
class A
{ public:
        A()
        {  cout<<"Constructor A called"<<endl;   }
        ~A()
        {  cout<<"Destructor A called"<<endl;  }
};
class B
{ public:
        B()
        {  cout<<"Constructor B called"<<endl;   }
        ~B()
        {  cout<<"Destructor B called"<<endl;  }
};
class C:virtual public A
{ public:
        C()
        {  cout<<"Constructor C called "<<endl;  }
        ~C()
        {  cout<<"Destructor C called"<<endl;  }};
```

Inheritance

```
class D:virtual public A
{ public:
            D()
            {  cout<<"Constructor D called "<<endl;  }
            ~D()
            {  cout<<"Destructor D called"<<endl;  }
};
class E
{ public:
            E()
            {  cout<<"Constructor E called "<<endl;  }
            ~E()
            {  cout<<"Destructor E called"<<endl;  }
};
class F:public B,public C,public D
{
  private:
            E Evar;
  public:
            F()
            {  cout<<"Constructor F called "<<endl;  }
```

Inheritance

```
~F()
{  cout<<"Destructor F called"<<endl;  }
};
int main()
{ F obj;
  cout<<"program end"<<endl;
}
```

Output:

```
Constructor A called
Constructor B called
Constructor C called
Constructor D called
Constructor E called
Constructor F called
program end
Destructor F called
Destructor E called
Destructor D called
Destructor C called
Destructor B called
Destructor A called
```

Discussions