INHERITACE

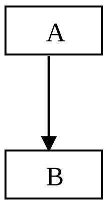
Inheritance is the process of creating new classes from an existing class. The existing class is known as the base class and newly created class is called as derived class. The derived class inherits the data members and member functions of the base class automatically, and also add some new member data and member function from its own.

Defining derived classes

```
 Class derived-class-name: visibility-made base-class-name
{
  access specifier:
       data-type member-data;
       member function;
};
```

Base Class Visibility	Derived class Visibility		
	Public Derivation	Private Derivation	Protected Derivation
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	protected
Public	Public	Private	protected

Single Inheritance



The process of creating a derived class from an existing base class is known as single inheritance.

Public Inheritance:

When the base class is publically inherited, public members of the base class become public members of the derived class and therefore they are accessible to the objects of the derived class.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a,b;
};
class B : public A
{
 public:
  int sum(int p,int q);
};
int B :: sum(int p,int q)
{
 int k;
 k=p+q;
 return(k);
}
void main()
{
 B ob1;
 clrscr();
 cin>>ob1.a>>ob1.b;
 cout<<ob1.sum(ob1.a,ob1.b);
 getch();
}
output:
10 20
30
```

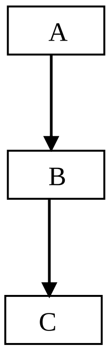
Private Inheritance:

When a base class is privately inherited by a derived class, public members of the base class become private members of the derived class and therefore the public members of the base class can only be accessed by the member functions of the derived class. They are inaccessible to the objects of the derived class

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a,b;
};
class B : private A
{
 public:
  int sum(void);
};
int B :: sum(void)
{
 int k;
 cin>>a>>b;
 k=a+b;
 return(k);
}
void main()
{
 B ob1;
 clrscr();
 cout<<ob1.sum();
 getch();
}
output:
10 20
30
```

Multilevel Inheritance:



The process of creating a derived class from another derived class is known as multilevel inheritance

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a,b;
  int sum(int p,int q);
};
class B : public A
{
 public:
  int sub(int p,int q);
};

class C : public B
{
 public:
  int mul(int p,int q);
};
class D : public C
{
 public:
  int div(int p,int q);
};
int A :: sum(int p,int q)
{
 int k;
 k=p+q;
 return(k);
}
int B :: sub(int p,int q)
{
 int k;
 k=p-q;
 return(k);
}
int C :: mul(int p,int q)
{
 int k;
 k=p*q;
 return(k);
}
int D :: div(int p,int q)
{
 int k;
 k=p/q;
 return(k);
}
void main()
{
 D ob1;
 clrscr();
 cin>>ob1.a>>ob1.b;
 cout<<ob1.sum(ob1.a,ob1.b)<<endl;
 cout<<ob1.sub(ob1.a,ob1.b)<<endl;
 cout<<ob1.mul(ob1.a,ob1.b)<<endl;
 cout<<ob1.div(ob1.a,ob1.b)<<endl;
 getch();
}
output:
20 10
30
10
200
2
```

Protected access specifier:

The member of a class declared as protected can be accessed from the derived class, but cannot be accessed from anywhere in other classes or the main program. In short, protected members behave like public members when it comes to derived classes and private members with respect to the rest of the program.

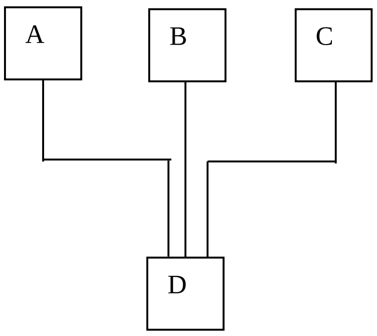
When a protected member is inherited in public mode, it becomes protected in the derived class too and therefore is accessible by the member functions of the derived class. It is also ready for further inheritance. A protected member, inherited in the private mode derivation, becomes private in the derived class. Although it is available to the member functions of the derived class, it is not available for further inheritance.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 protected:
  int a,b;
 public:
  void getdata(void);
  int sum(void);
};
class B : public A
{
 public:
  int sub(void);
};
class C : public B
{
 public:
  int mul(void);
};
class D : public C
{
 public:
  int div(void);
};
void A :: getdata(void)
{
 cin>>a>>b;
}
int A :: sum(void)
{
 int k;
 k=a+b;
 return(k);
}
int B :: sub(void)
{
 int k;
 k=a-b;
 return(k);
}
int C :: mul(void)
{
 int k;
 k=a*b;
 return(k);
}
int D :: div(void)
{
 int k;
 k=a/b;
 return(k);
}
void main()
{
 D ob1;
 clrscr();
 ob1.getdata();
 cout<<ob1.sum()<<endl;
 cout<<ob1.sub()<<endl;
 cout<<ob1.mul()<<endl;
 cout<<ob1.div()<<endl;
 getch();
}
```
output:

```
20 10
30
10
200
```

Multiple Inheritance

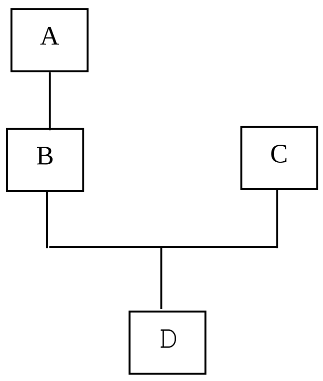


The process of creating a derived class from several base class is known as multiple inheritance.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a;
  int sum(int p,int q,int r);
};
class B
{
 public:
  int b;
  int mul(int p,int q,int r);
};
class C
{
 public:
  int c;
  int avg(int p,int q,int r);
};
class D: public A,public B,public C
{
 public:
  int big(int p,int q,int r);
};
int A :: sum(int p,int q,int r)
{
 int k;
 k=p+q+r;
 return(k);
}
int B :: mul(int p,int q,int r)
{
 int k;
 k=p*q*r;
 return(k);
}
int C :: avg(int p,int q,int r)
{
 int k;
 k=(p+q+r)/3;
 return(k);
}
int D :: big(int p,int q,int r)
{
 int k;
 if(p>q)
 {
  if(p>r)
   k=p;
  else
   k=r;
 }
 else
 {
  if(q>r)
   k=q;
  else
   k=r;
 }
 return(k);
}
void main()
{
 D ob1;
 clrscr();
 cin>>ob1.a>>ob1.b>>ob1.c;
 cout<<ob1.sum(ob1.a,ob1.b,ob1.c)<<endl;
 cout<<ob1.mul(ob1.a,ob1.b,ob1.c)<<endl;
 cout<<ob1.avg(ob1.a,ob1.b,ob1.c)<<endl;
 cout<<ob1.big(ob1.a,ob1.b,ob1.c)<<endl;
 getch();
}
output:
30 20 10
60
6000
20
30
```

Hybrid Inheritance:

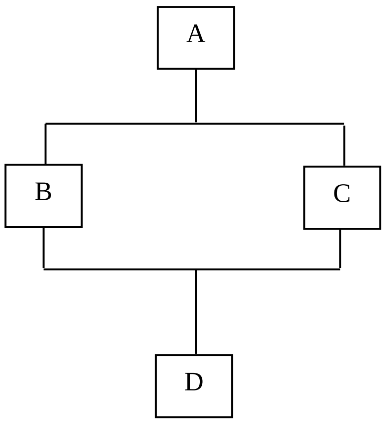


Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a,b,c;
};
class B : public A
{
 public:
  int sum(int p,int q,int r);
};
class C
{
 public:
  int mul(int p,int q,int r);
};
class D: public B,public C
{
 public:
  int avg(int p,int q,int r);
};
int B :: sum(int p,int q,int r)
{
 int k;
 k=p+q+r;
 return(k);
}
int C :: mul(int p,int q,int r)
{
 int k;
 k=p*q*r;
 return(k);
}
int D :: avg(int p,int q,int r)
{
 int k;
 k=(p+q+r)/3;
 return(k);
}
void main()
{
 D ob1;
 clrscr();
 cin>>ob1.a>>ob1.b>>ob1.c;
 cout<<ob1.sum(ob1.a,ob1.b,ob1.c)<<endl;
 cout<<ob1.mul(ob1.a,ob1.b,ob1.c)<<endl;
 cout<<ob1.avg(ob1.a,ob1.b,ob1.c)<<endl;
 getch();
}
```

```
output:
30 20 10
60
6000
20
```

Virtual base class



A base class that has been qualified as virtual in the inheritance definition. In multiple inheritance, a derived class can inherit the members of a base class via two or more inheritance paths. If the base class is not virtual that derived class will inherit more than one copy of the members of the base class for a virtual base class, however, only one copy of its members will be inherited regardless of the number of inheritance paths between the base class and the derived class.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  int a,b,c;
  int sum(int p,int q);
};
class B : virtual public A
{
 public:
  int mul(int p,int q);
};
class C : virtual public A
{
 public:
  int div(int p,int q);
};
class D : public B,public C
{
 public:
  int diff(int p,int q);
};
int A :: sum(int p,int q)
{
 return(p+q);
}
int B :: mul(int p,int q)
{
 return(p*q);
}
int C :: div(int p,int q)
{
 return(p/q);
}
int D :: diff(int p,int q)
{
 return(p-q);
}
void main()
{
 D obj;
 clrscr();
 cin>>obj.a>>obj.b;
 cout<<obj.sum(obj.a,obj.b)<<endl;
 cout<<obj.mul(obj.a,obj.b)<<endl;
 cout<<obj.div(obj.a,obj.b)<<endl;
 cout<<obj.diff(obj.a,obj.b)<<endl;
 getch();
}
output:
20 10
30
200
2
10
```

Constructors in derived classes

In case of single inheritance, the base constructor is executed first and then the constructor in the derived class is executed.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
};
class B : public A
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a*b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 B obj;
 getch();
}
output:
20 10
30
20 5
100
```

In case of multilevel inheritance, the constructors will be executed in the order of inheritance.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
};
class B : public A
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a*b;
   cout<<c<<endl;
  }
};
class C : public B
{
 public:
  C()
  {
   int a,b,c;
   cin>>a>>b;
   c=a-b;
   cout<<c<<endl;
  }
};
class D : public C
{
 public:
  D()
  {
   int a,b,c;
   cin>>a>>b;
   c=a/b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 D obj;
 getch();
}
output:
20 10
30
20 10
200
20 10
10
20 10
```

2

In case of multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class.

Program:
```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
};
class B
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a*b;
   cout<<c<<endl;
  }
};
class C
{
 public:
  C()
  {
   int a,b,c;
   cin>>a>>b;
   c=a-b;
   cout<<c<<endl;
  }
};
class D : public A,public B,public C
{
 public:
  D()
  {
   int a,b,c;
   cin>>a>>b;
   c=a/b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 D obj;
 getch();
}
output:
```

```
20 10
30
20 10
200
20 10
10
20 10
2
```

Constructors in derived classes with arguments:

The constructor of the derived class receives the entire list of values as its arguments and passes them on to the base constructors in the order in which they are declared in the derived class. The base constructors are called and executed before executing the statements in the body of the derived constructors.

The general form of defining a derived constructor is

```
Derived_construct(arglist1,arglist2,……………..):
base1(arglist1),base2(arglist2)……..
{
  Body of derived constructor
}
```

The header line of derived constructor function contains two parts separated by a colon (:). The first part provides the declaration of the arguments that are passed to the derived constructor and the second part lists the function calls to the base constructors.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A(int a,int b)
  {
   int c;
   c=a+b;
   cout<<c<<endl;
  }
};
class B : public A
{
 public:
  B(int a,int b) : A(a,b)
  {
   int c;
   c=a*b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 B obj(5,6);
 getch();
}
output:
11
30
```

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A(int a,int b)
  {
   int c;
   c=a+b;
   cout<<c<<endl;
  }
};
class B : public A
{
 public:
  B(int a,int b) : A(a,b)
  {
   int c;
   c=a*b;
   cout<<c<<endl;
  }
};
class C : public B
{
 public:
  C(int a,int b) : B(a,b)
  {
   int c;
   c=a-b;
   cout<<c<<endl;
  }
};
class D : public C
{
 public:
  D(int a,int b) : C(a,b)
  {
   int c;
   c=a/b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 D obj(5,6);
 getch();
}
```

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A(int a,int b)
  {
   int c;
   c=a+b;
   cout<<c<<endl;
  }
};
class B
{
 public:
  B(int a,int b)
  {
   int c;
   c=a*b;
   cout<<c<<endl;
  }
};
class C
{
 public:
  C(int a,int b)
  {
   int c;
   c=a-b;
   cout<<c<<endl;
  }
};
class D : public A,public B,public C
{
 public:
  D(int a,int b) : A(a,b),B(a,b),C(a,b)
  {
   int c;
   c=a/b;
   cout<<c<<endl;
  }
};
void main()
{
 clrscr();
 D obj(5,6);
 getch();
}
output:
11
30
-1
0
```

Destructors in derived class:

In case of single inheritance, the derived destructor is executed first and then the destructor in the base class is executed.

Program:

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

```
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
  ~A()
  {
   cout<<"End of class A"<<endl;
  }
};
class B : public A
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a-b;
   cout<<c<<endl;
  }
  ~B()
  {
   cout<<"End of class B"<<endl;
  }
};
void main()
{
 clrscr();
 B obj;
 getch();
}
output:
10 20
30
10 20
-10
End of class B
End of class A
```

In case of multilevel inheritance, the destructors will be executed in the reverse order of inheritance.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
  ~A()
  {
   cout<<"End of class A"<<endl;
  }
};
class B : public A
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a-b;
   cout<<c<<endl;
  }
  ~B()
  {
   cout<<"End of class B"<<endl;
  }
};
class C: public B
{
 public:
  C()
  {
   int a,b,c;
   cin>>a>>b;
   c=a*b;
   cout<<c<<endl;
  }
  ~C()
  {
   cout<<"End of class C"<<endl;
  }
};
class D : public C
{
 public:
  D()
  {
   int a,b,c;
   cin>>a>>b;
   c=a/b;
   cout<<c<<endl;
  }
  ~D()
  {
   cout<<"End of class D"<<endl;
  }
};
void main()
{
 clrscr();
 D obj;
 getch();
}
output:
20 10
30
20 10
10
20 10
```

```
200
20 10
10
End of class D
End of class C
End of class B
End of class A
```

In case of multiple inheritance the derived class destructor will be executed first. After that the base class destructors are executed in the reverse order in which they appear in the declaration of the derived class.

Program:

```
#include<iostream.h>
#include<conio.h>
class A
{
 public:
  A()
  {
   int a,b,c;
   cin>>a>>b;
   c=a+b;
   cout<<c<<endl;
  }
  ~A()
  {
   cout<<"End of class A"<<endl;
  }
};
class B
{
 public:
  B()
  {
   int a,b,c;
   cin>>a>>b;
   c=a-b;
   cout<<c<<endl;
  }
  ~B()
  {
   cout<<"End of class B"<<endl;
  }
};
class C
{
 public:
  C()
  {
   int a,b,c;
   cin>>a>>b;
   c=a*b;
   cout<<c<<endl;
  }
  ~C()
  {
   cout<<"End of class C"<<endl;
  }
};
class D : public A,public B,public C
{
 public:
  D()
  {
   int a,b,c;
   cin>>a>>b;
   c=a/b;
   cout<<c<<endl;
  }
  ~D()
  {
   cout<<"End of class D"<<endl;
  } };
void main()
{
 clrscr();
 D obj;
 getch();
}
output:
20 10
30
20 10
10
20 10
200
20 10
10
End of class D
End of class C
End of class B
End of class A
```