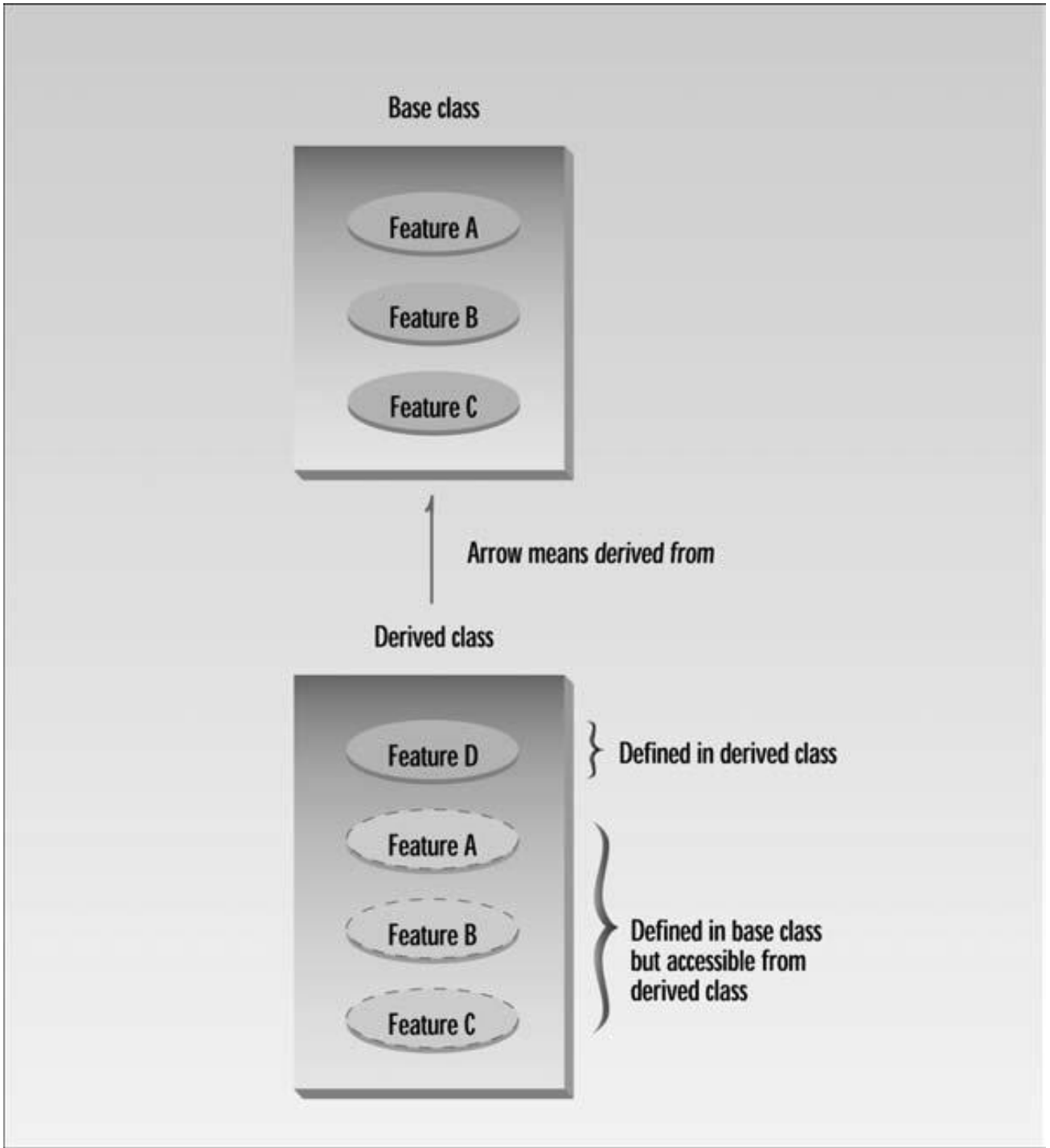
Inheritance and Polymorphism

Contents

- Inheritance Concepts
- Inheritance Examples
- Implementing Inheritance in C++
- Polymorphism
- Dynamic Binding
- Virtual Function Examples


Base class
Feature A
Feature B
Feature C
Arrow means derived from
Derived class
Feature D
Feature A
Feature B
Feature C
Defined in derived class
Defined in base class
but accessible from
derived class

Inheritance Concepts

- Deriving a new class (subclass) from an existing class (base class or superclass).

- Inheritance creates a hierarchy of related classes which share code and interface.

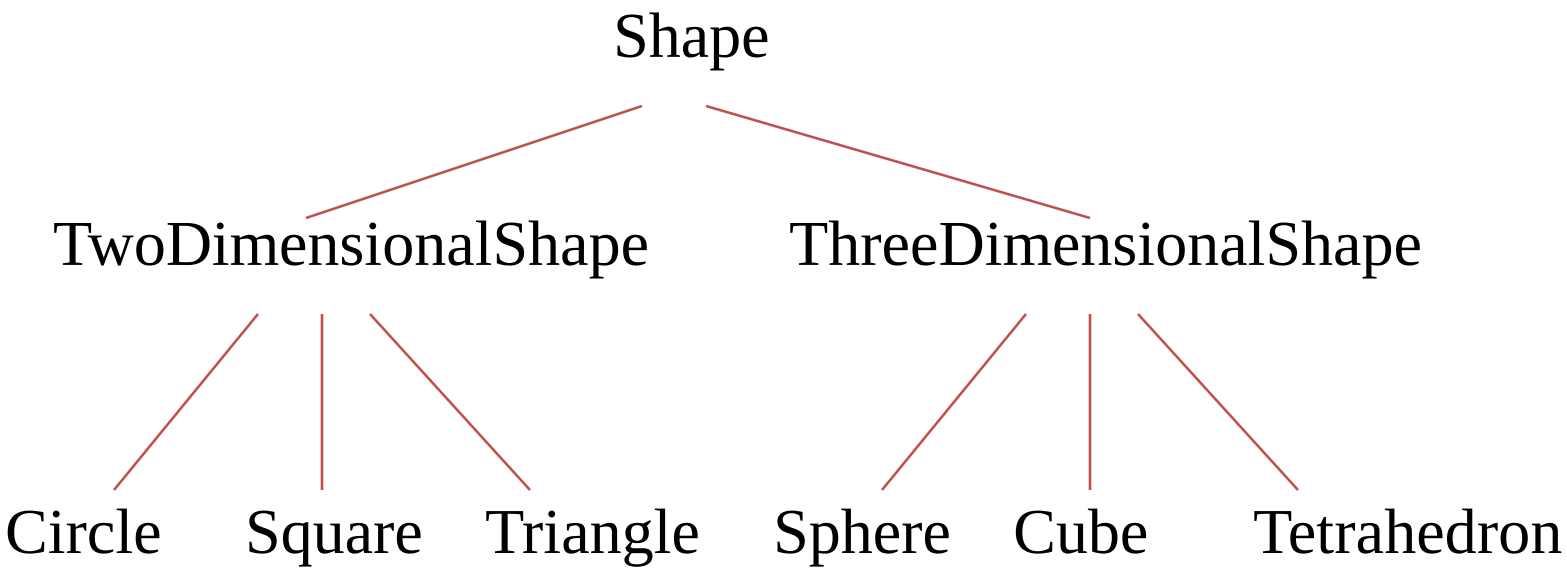
Inheritance Examples

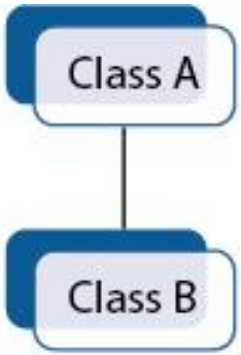
Base Class	Derived Classes
Shape	Circle Triangle Rectangle
Loan	CarLoan HomeLoan Loan

More Examples

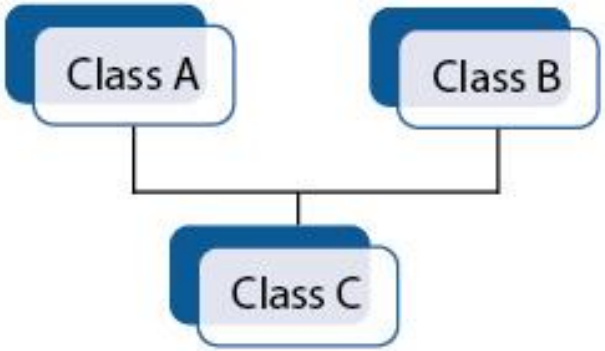
Base Class	Derived Classes
Employee	Manager Researcher Worker
Account	CheckingAccount SavingAccount

Shape class hierarchy

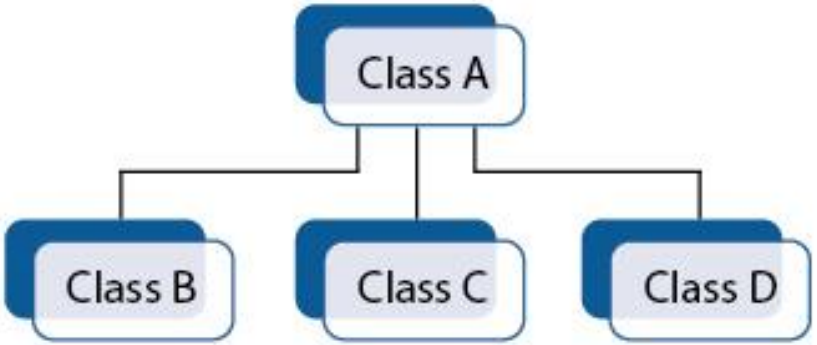




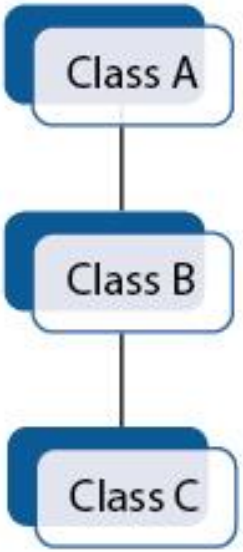
Single Inheritance



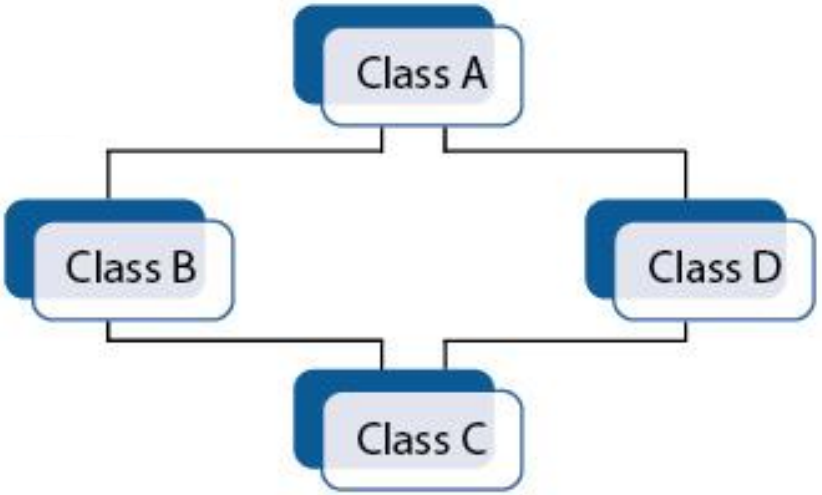
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

Implementing Inheritance in C++

- Develop a base class.
- Use it to define a derived class.

Syntax:

```
class derived_class_name : access_mode base_class_name
{
        // body of class
};
```

Base class member visibility	Type of inheritance		
	public	private	protected
public	public	private	protected
private	Not inherited	Not inherited	Not inherited
protected	protected	private	protected

Example-1: Public Inheritance

```
class base
{
    int i, j;
 public:
    void set_ij(int a, int b)
    { i=a; j=b; }

    void show_ij()
    { cout << i << " " << j << "\n"; }
};
```

```
class derived : public base
{
    int k;
public:
    void set_k( int val)
    {   k = val;    }

    void show_k()
    {   cout<<k; }
};
```

```
int main()
{
    derived ob;
    ob.set_ij( 5, 10 );
    ob.set_k( 20 );
    ob.show_ij();
    ob.show_k();
}
```

OUTPUT:

```
5 10
20
```

Example-2: Private Inheritance

```
3   class base
4   {
5       int i, j;
6    public:
7       void set_ij(int a, int b)
8       { i=a; j=b; }
9
10      void show_ij()
11      { cout << i << " " << j << "\n"; }
12  };
```

```
13  class derived : private base
14  {
15      int k;
16  public:
17      void set_k( int val)
18      {   k = val;    }
19
20      void show_k()
21      {   cout<<k; }
22  };
```

```
23  int main()
24  {
25      derived ob;
26      //ob.set_ij( 5, 10 ); Error
27      ob.set_k( 20 );
28      //ob.show_ij(); Error
29      ob.show_k();
30  }
```

Example-3: protected members

```
3   class base
4   {
5    protected:
6       int i, j;
7    public:
8       void set_ij(int a, int b)
9       { i=a; j=b; }
10
11      void show_ij()
12      { cout << i << " " << j << "\n"; }
13  };
14  class derived : public base
15  {
16      int k;
17  public:
18      void set_k()
19      {   k = i*j;    }
20
21      void show_k()
22      {   cout<<k; }
23  };
```

```
24  int main()
25  {
26      derived ob;
27      ob.set_ij( 5, 10 );
28      ob.set_k( );
29      ob.show_ij();
30      ob.show_k();
31  }
```

OUTPUT:

```
5 10
50
```

Example-4: protected members

```
3   class base
4   {
5    protected:
6       int i, j;
7    public:
8       void set_ij(int a, int b)
9       { i=a; j=b; }
10
11      void show_ij()
12      { cout << i << " " << j << "\n"; }
13  };
14  class derived : base
15  {
16      int k;
17  public:
18      void set_k()
19      {   k = i*j;    }
20
21      void show_k()
22      {   cout<<k; }
23  };
```

```
24  int main()
25  {
26      derived ob;
27      //ob.set_ij( 5, 10 ); Error
28      ob.set_k( );
29      //ob.show_ij(); Error
30      ob.show_k();
31  }
```

Question - 1

```
class base
{
public:
    void f1()
    {
        cout<<"Hello";
    }
};

class der1 : public base
{
public:
    void f2()
    {
        cout<<"\n Hi..";
    }
};
```

```
class der2 : base
{
public:
    void f3()
    {
        cout<<"\n goodbye..";
    }
};
```

```
int main()
{
    der1 d1;
    der2 d2;
    d1.f1();
    d2.f1();    //Error !
    return 0;
}
```

Question - 2

```
class A
{
private:
    int privdataA;
protected:
    int protdataA;
public:
    int pubdataA;
};
```

```
class B : public A
{
public:
    void test1()
    {
        int a;
        a = privdataA; //error:
        a = protdataA;
        a = pubdataA;
    }
};
```

```
class C : private A
{
public:
    void test2()
    {
        int a;
        a = privdataA; //error:
        a = protdataA;
        a = pubdataA;
    }
};
```

```
34  int main()
35  {
36      int a;
37      B objB;
38      a = objB.privdataA;  //error: not accessible
39      a = objB.protdataA;  //error: not accessible
40      a = objB.pubdataA;   //OK (A public to B)
41      C objC;
42      a = objC.privdataA;  //error: not accessible
43      a = objC.protdataA;  //error: not accessible
44      a = objC.pubdataA;   //error: not accessible
45      return 0;
46  }
```

Example: Multiple inheritance

```
3   class base1
4   {
5   protected:
6       int x;
7   public:
8       void showx() { cout << x << "\n"; }
9   };
10
11  class base2
12  {
13  protected:
14      int y;
15  public:
16  void showy() {cout << y << "\n";}
17  };
19  class derived: public base1, public base2
20  {
21  public:
22      void set(int i, int j) { x=i; y=j; }
23  };
```

```
24  int main()
25  {
26      derived ob;
27      ob.set(10, 20);
28      ob.showx();
29      ob.showy();
30      return 0;
31  }
```

Example: Multilevel inheritance

```
class BASE
{
protected:
    int i, j;
public:
    void set(int a, int b) { i = a; j=b; }
    void show() { cout << i << j <<"\n"; }
};
class derived1 : public BASE
{
    int k;
public:
    void setk() { k = i*j; }
    void showk() { cout <<k<<"\n"; }
};

class derived2 : public derived1
{
    int m;
public:
    void setm() { m = i-j; }
    void showm() { cout <<m<<"\n"; }
};
```

OUTPUT:

```
2 3
6
3 4
12
-1
```

```
int main()
{
    derived1 ob1;
    derived2 ob2;
    ob1.set(2, 3);
    ob1.show();
    ob1.setk();
    ob1.showk();
    ob2.set(3, 4);
    ob2.show();
    ob2.setk();
    ob2.setm();
    ob2.showk();
    ob2.showm();
    return 0;
}
```

```
3  class BASE
4  {
5  protected:
6      int i, j;
7  public:
8      void set(int a, int b) { i = a; j=b; }
9      void show() { cout << i <<" "<< j <<"\n"; }
10 };
11 class derived1 : private BASE
12 {
13     int k;
14 public:
15     void setk() { k = i*j; }
16     void showk() { cout <<k<<"\n"; }
17 };
18 class derived2 : public derived1
19 {
20     int m;
21 public:
22     void setm() { m = i-j; }
23     void showm() { cout <<m<<"\n"; }
24 };
```

```
class BASE
{
protected:
    int i, j;
public:
    void set(int a, int b) { i = a; j=b; }
    void show() { cout << i <<" "<< j <<"\n"; }
};
class derived1 : private BASE
{
    int k;
public:
    void setk() { k = i*j; }
    void showk() { cout <<k<<"\n"; }
};
```

```
class derived2 : public derived1
{
    int m;
public:
    void setm() { m = i-j; }        // Error
    void showm() { cout <<m<<"\n"; }
};
```

```
int main()
{
    derived1 ob1;
    derived2 ob2;
    ob1.set(1, 2);
    ob1.show();
    ob2.set(3, 4);
    ob2.show();
    return 0;
}
```

Inheritance & Constructors

```
3   class base
4   {
5     public:
6       base()
7       { cout<<"base-class constr.."; }
8   };
9   class derived : public base
10  {
11    public:
12      derived()
13      { cout<<"\nderived-class constr.."; }
14  };
```

```
15  int main()
16  {
17      derived ob;
18  }
```

```
base-class constr..
derived-class constr..
```

Inheritance with Constructors & Destructor

```
3   class base
4   {
5     public:
6       base()
7       { cout<<"base-class constr.."; }
8       ~base()
9       { cout<<"\nbase-class destr..";}
10  };
11  class derived : public base
12  {
13    public:
14      derived()
15      { cout<<"\nderived-class constr.."; }
16
17      ~derived()
18      { cout<<"\nderived-class destr.."; }
19  };
20  int main()
21  {
22      derived ob;
23  }
```

```
base-class constr..
derived-class constr..
derived-class destr..
base-class destr..
```

Question-1:

```
3   class base
4   {
5   public:
6       base() { cout << "Constructor of base\n"; }
7       ~base() { cout << "Destructor of base\n"; }
8   };
9
10  class derived1 : public base
11  {
12  public:
13      derived1() { cout << "Constructor of derived1\n"; }
14      ~derived1() { cout << "Destructor of derived1\n"; }
15  };
17  class derived2: public derived1
18  {
19  public:
20      derived2() { cout << "Constructor of derived2\n"; }
21      ~derived2() { cout << "Destructor of derived2\n"; }
22  };
```

```
24  int main()
25  {
26      derived2 ob;
27      return 0;
28  }
```

OUTPUT:

```
Constructor of base
Constructor of derived1
Constructor of derived2
Destructor of derived2
Destructor of derived1
Destructor of base
```

Question-2:

```
class base1
{
public:
    base1()
    {
        cout << "\nConstructor of base1\n";
    }
};
class base2
{
public:
    base2()
    {
        cout<<"\nConstructor base2";
    }
};
class derived : public base2, public base1
{
public:
    derived()
    {
        cout<<"\nConstructor of derived";
    }
};
```

```
int main()
{
    derived ob;
    return 0;
}
```

```
Constructor base2
Constructor of base1

Constructor of derived
```

Method of Inheritance	Order of Execution

`class D: public B`
`{`
`...`
`};`
	B( ): base constructor
	D( ): derived constructor

`class D: public B1, public B2`
`{`
`...`
`};`
	B1( ): base constructor
	B2( ): base constructor
	D( ): derived constructor

`class D: public B1, virtual B2`
`{`
`..`
`};`
	B2( ): virtual base constructor
	B1( ): base constructor
	D( ): derived constructor

`class D1: public B`
`{`
`...`
`};`
`class D2: public D1`
`{`
`..`
`};`
	B( ): super base constructor
	D1( ): base constructor
	D2( ): derived constructor

Table 14.3: Order of invocation of constructors

Question 1

```
class base
{
public:
    base() { cout << "Constructor of base\n"; }
};

class derived : public base
{
    int a;
public:
    derived() { cout <<"Default Constructor of derived\n"; }
    derived(int x)
    { a = x; cout << "1-arg Constructor of derived\n"; }
};
```

```
int main()
{
    derived ob(10);
    return 0;
}
```

OUTPUT:

```
Constructor of base
1-arg Constructor of derived
```

Question 2

```
class base
{
public:
    base()
    {
        cout<<"\n Constructor of base\n";
    }
    ~base()
    {
        cout<<"\n Destr. of base";
    }
};
```

```
class derived : base
{
public:
    derived() { cout<<"\n Def.Constructor of derived"; }

    derived( int x ) { cout<<"\ 1-arg.Constructor of derived"; }

    ~derived() { cout<<"\n Destr. of derived"; }
};
```

```
int main()
{
    derived ob1(10), ob2;
    return 0;
}
```

OUTPUT:

```
Constructor of base
1-arg.Constructor of derived
Constructor of base

Def.Constructor of derived
Destr. of derived
Destr. of base
Destr. of derived
Destr. of base
```

Passing Parameters to Base-Class Constructors

```
class base1
{
public:
    base1( args )
    {
     . . .
    }
};
```

```
class base2
{
 public:
   base2( args )
    {
      . . .
    }
};
```

```
class base3
{
 public:
   base3( args )
    {
      . . .
    }
};
```

```
Derived_constr(arg_list) : base1 ( arglist ), base2 ( arglist ), base3 ( arglist )
{
    // body of derived constructor
}
```

Example: passing parameters to base-class constructor

```
3   class base
4   {
5   protected:
6       int base_a;
7   public:
8       base( int x )
9       {
10          base_a = x;
11      }
12  };
```

```
13  class derived : public base
14  {
15      int der_a;
16  public:
17      derived( int m, int n ) : base(m)
18      {
19          der_a = n;
20      }
21      void display()
22      {
23          cout<<" base_a="<<base_a
24              <<"\n der_a="<<der_a;
25      }
26  };
```

```
27  int main()
28  {
29      derived D( 10 , 20 );
30      D.display();
31      return 0;
32  }
```

Overriding Member Functions

```
3   class base
4   {
5   public:
6       base()
7       {
8           cout << "\nConstructor of base\n";
9       }
10      void show()
11      {
12          cout<<"\n Show() of base.";
13      }
14  };
```

```
28  int main()
29  {
30      derived ob;
31      ob.show();
32      return 0;
33  }
```

```
16  class derived : public  base
17  {
18  public:
19      derived()
20      {
21          cout<<"\nConstructor of derived";
22      }
23      void show()
24      {
25          cout<<"\n Show() of derived.";
26      }
27  };
```

```
3   class base
4   {
5   public:
6       void show()
7       {
8           cout<<"\n show() of base\n";
9       }
10  };
```

```
12  class derived:public base
13  {
14      public:
15          void show()
16          {
17              //base::show();
18              cout<<"\n show() of derived\n";
19          }
20  };
```

```
21  int main()
22  {
23      derived d;
24      d.base::show();
25      d.show();
26      return 0;
27  }
```

```
class base1
{
public:
    void show()
    {
        cout<<"\n show() of base1\n";
    }
};
class base2
{
public:
    void show()
    {
        cout<<"\n show() of base2\n";
    }
};
class derived:public base1, public base2
{
    public:
        void show()
        {
            cout<<"\n show() of derived.\n";
        }
};
```

```
int main()
{
    derived d;
    d.show();
    return 0;
}
```

```
show() of derived.
```

```
3   class base1
4   {
5   public:
6       void show()
7       {
8           cout<<"\n show() of base1\n";
9       }
10  };
11  class base2
12  {
13  public:
14      void show()
15      {
16          cout<<"\n show() of base2\n";
17      }
18  };
19  class derived:public base1, public base2
20  {
21  };
```

```
22  int main()
23  {
24      derived d;
25      d.show();
26      return 0;
27  }
```

call to show() is ambiguous !!!

```
3   class base1
4   {
5   public:
6       void show()
7       {
8           cout<<"\n show() of base1\n";
9       }
10  };
11  class base2
12  {
13  public:
14      void show()
15      {
16          cout<<"\n show() of base2\n";
17      }
18  };
19  class derived:public base1, public base2
20  {
21  };
```

```
22  int main()
23  {
24      derived d;
25      d.base1::show();
26      d.base2::show();
27      return 0;
28  }
```

Question

```
class base
{
    int b1, b2;
public:
    // constructor
};
class derived : public base
{
    int d1, d2, d3;
public:
    // constructor
};
```

```
int main()
{
    derived D( 10 , 20, 30, 40, 50 );
    D.display();
}
```

```
3   class base
4   {
5       int b1, b2;
6   public:
7       base( int x, int y )
8       {
9           b1 = x; b2 = y;
10      }
11      void display()
12      {
13          cout<<" b1:"<<b1
14              <<"\n b2:"<<b2;
15      }
16  };
```

```
17  class derived : public base
18  {
19      int d1, d2, d3;
20  public:
21      derived( int a, int b, int c, int d, int e ) : base(b,d)
22      {
23          d1 = a; d2 = c; d3 = e;
24      }
25      void display()
26      {
27          base::display();
28          cout<<"\n d1:"<<d1<<"\n d2:"<<d2<<"\n d3:"<<d3;
29      }
30  };
31  int main()
32  {
33      derived D( 10 , 20, 30, 40, 50 );
34      D.display();
35  }
```

```
b1:20
b2:40
d1:10
d2:30
d3:50
```

Function overriding: Stack Example

```
3   const int MAX = 3;
4   class Stack
5   {
6   protected:
7       int st[MAX];
8       int top;
9   public:
10      Stack()
11      { top = -1; }
12
13      void push(int var)
14      { st[++top] = var; }
15
16      int pop()
17      { return st[top--]; }
18  };
```

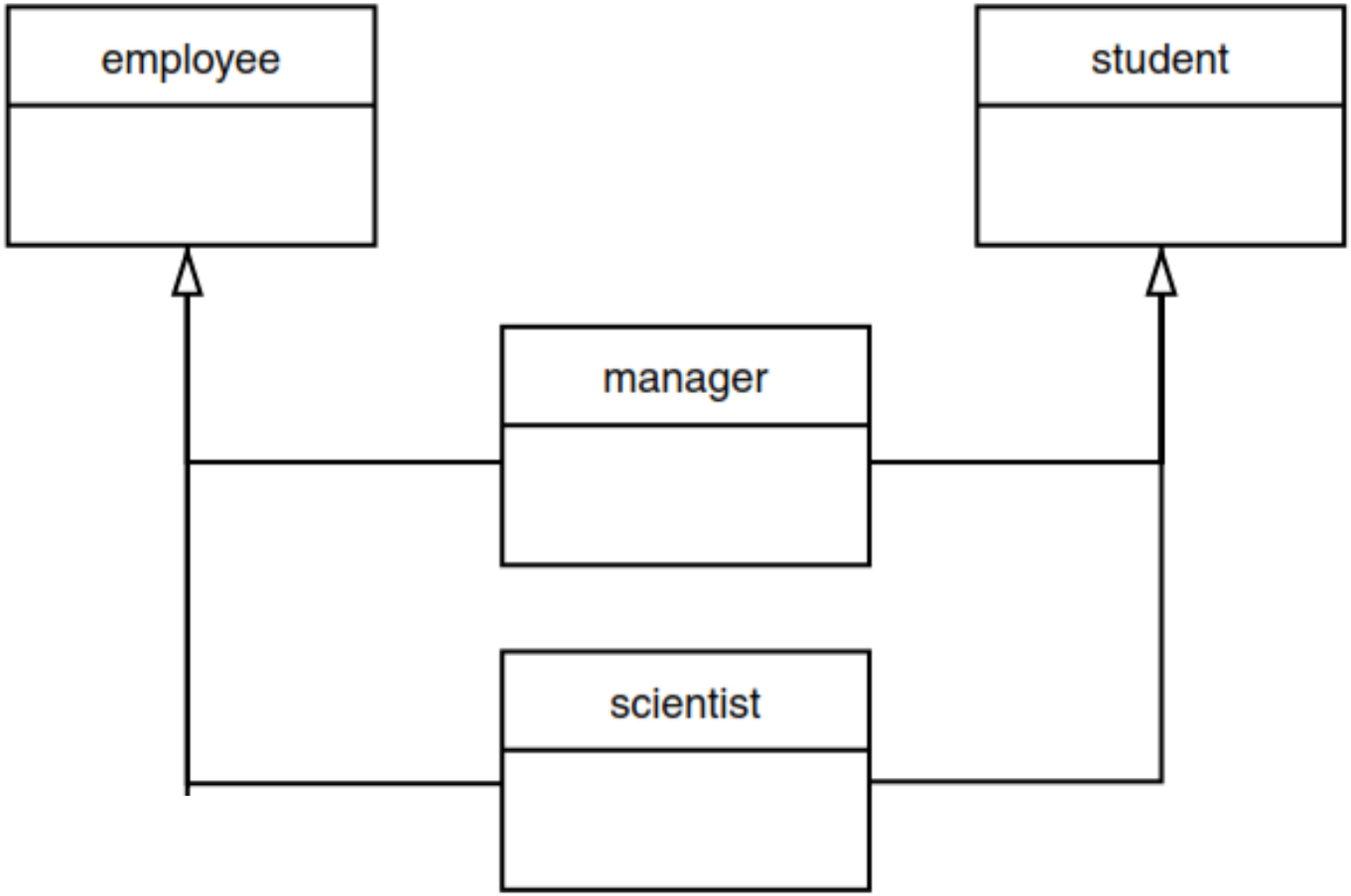
```
20  class Stack2 : public Stack
21  {
22  public:
23      void push(int var)
24      {
25          if(top >= MAX-1)
26          { cout << "\nError: stack is full";
27            exit(1);
28          }
29            Stack::push(var);
30      }
31      int pop()
32      {
33          if(top < 0)
34          { cout << "\nError: stack is empty\n";
35          exit(1);
36          }
37          return Stack::pop();
38      }
39  };
```

```
40  int main()
41  {
42      Stack2 s1;
43      s1.push(11);
44      s1.push(22);
45      s1.push(33);
46      cout <<"\n"<< s1.pop();
47      cout <<"\n"<< s1.pop();
48      cout <<"\n"<< s1.pop();
49      cout <<"\n"<< s1.pop();
50      return 0;
51  }
```

OUTPUT

```
33
22
11
Error: stack is empty
```

Multiple inheritance



```
class student
   { };
class employee
   { };
class manager : private employee, private student
   { };
class scientist : private employee, private student
   { };
```

```
class student                    //educational background
   {
   private:
      char school[LEN];          //name of school or university
      char degree[LEN];          //highest degree earned
   public:
      void getedu()
         {
         cout << "   Enter name of school or university: ";
         cin >> school;
         cout << "   Enter highest degree earned \n";
         cout << "   (Highschool, Bachelor's, Master's, PhD): ";
         cin >> degree;
         }

      void putedu() const
         {
         cout << "\n   School or university: " << school;
         cout << "\n   Highest degree earned: " << degree;
         }
   };
```

```
class employee
   {
   private:
      char name[LEN];          //employee name
      unsigned long number;    //employee number

   public:
      void getdata()
         {
         cout << "\n   Enter last name: "; cin >> name;
         cout << "   Enter number: ";      cin >> number;
         }
      void putdata() const
         {
         cout << "\n   Name: " << name;
         cout << "\n   Number: " << number;
         }
   };
```

```
class manager : private employee, private student  //management
   {
   private:
      char title[LEN];          //"vice-president" etc.
      double dues;              //golf club dues
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter title: ";          cin >> title;
         cout << "   Enter golf club dues: "; cin >> dues;
         student::getedu();
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Title: " << title;
         cout << "\n   Golf club dues: " << dues;
         student::putedu();
         }
   };
```

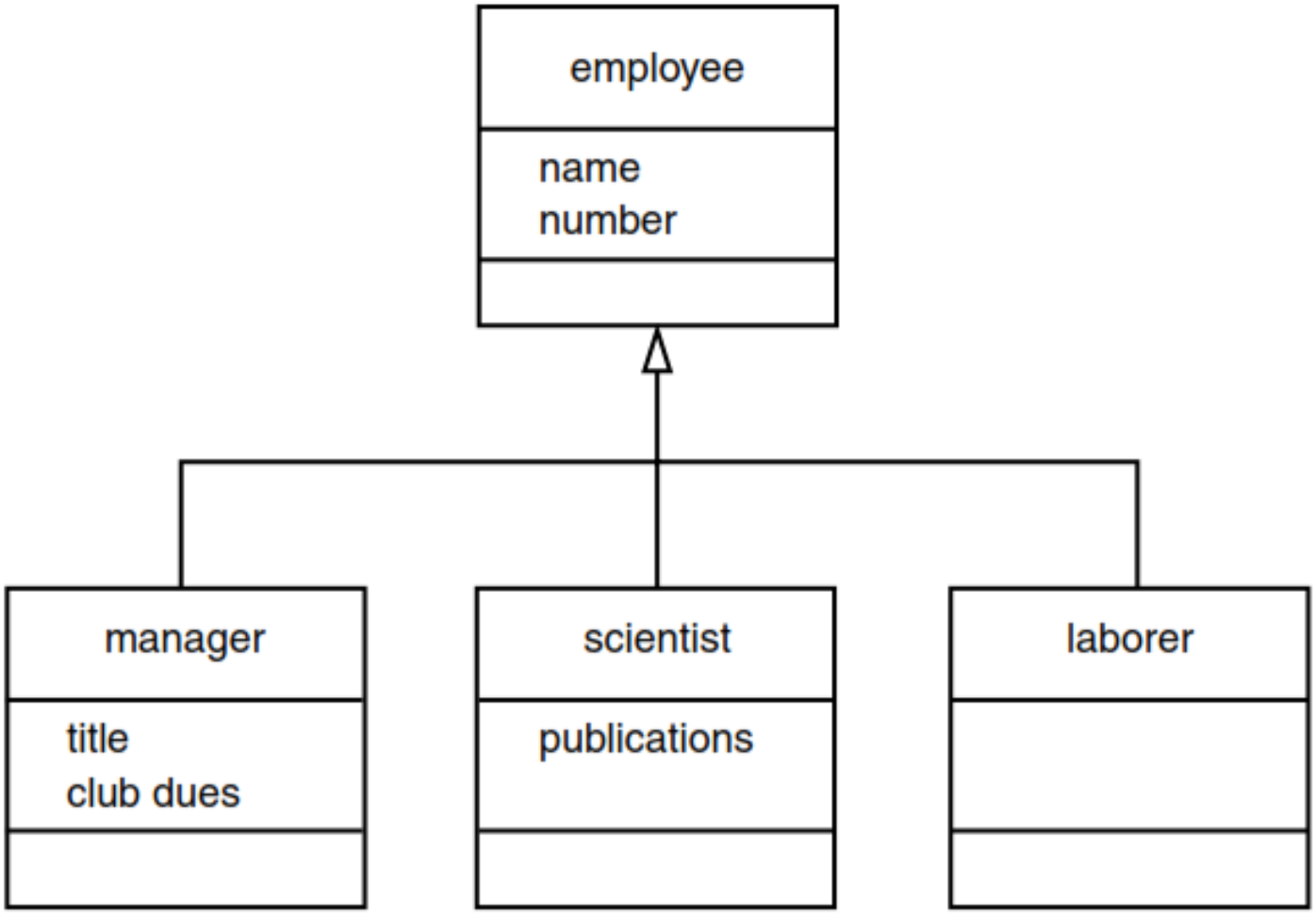
```
class scientist : private employee, private student  //scientist
   {
   private:
      int pubs;     //number of publications
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         student::getedu();
         }

      void putdata() const
         {
         employee::putdata();
         cout << "\n   Number of publications: " << pubs;
         student::putedu();
         }
   };
```

```
int main()
   {
   manager m1;
   scientist s1, s2;
   laborer l1;

   cout << endl;
   cout << "\nEnter data for manager 1";    //get data for
   m1.getdata();                            //several employees

   cout << "\nEnter data for scientist 1";
   s1.getdata();

   cout << "\nEnter data for scientist 2";
   s2.getdata();

   cout << "\nData on manager 1";
   m1.putdata();
   cout << "\nData on scientist 1";
   s1.putdata();
   }
```

Output

```
Enter data for manager 1
   Enter last name: Bradley
   Enter number: 12
   Enter title: Vice-President
   Enter golf club dues: 100000
   Enter name of school or university: Yale
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): Bachelor's

Enter data for scientist 1
   Enter last name: Twilling
   Enter number: 764
   Enter number of pubs: 99
   Enter name of school or university: MIT
   Enter highest degree earned
   (Highschool, Bachelor's, Master's, PhD): PhD
```

Class Hierarchies



```
class employee                          //employee class
   {
   private:
      char name[LEN];                   //employee name
      unsigned long number;             //employee number
   public:
      void getdata()
         {
         cout << "\n   Enter last name: "; cin >> name;
         cout << "   Enter number: ";      cin >> number;
         }


      void putdata() const
         {
         cout << "\n   Name: " << name;
         cout << "\n   Number: " << number;
         }
   };
```

```
class manager : public employee    //management class
   {
   private:
      char title[LEN];             //"vice-president" etc.
      double dues;                 //golf club dues
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter title: ";          cin >> title;
         cout << "   Enter golf club dues: "; cin >> dues;
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Title: " << title;
         cout << "\n   Golf club dues: " << dues;
         }
   };
```

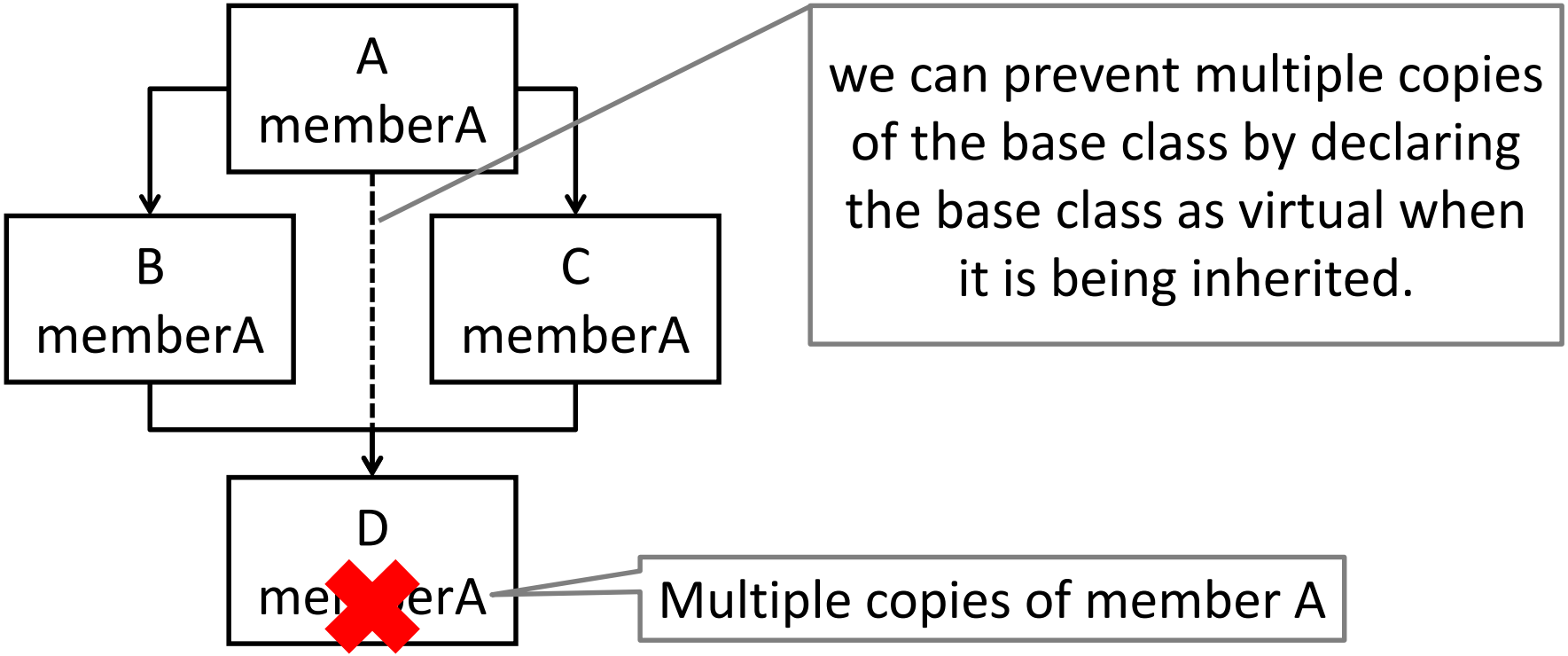
```
class scientist : public employee  //scientist class
   {
   private:
      int pubs;                        //number of publications
   public:
      void getdata()
         {
         employee::getdata();
         cout << "   Enter number of pubs: "; cin >> pubs;
         }
      void putdata() const
         {
         employee::putdata();
         cout << "\n   Number of publications: " << pubs;
         }
   };
```

```
class  laborer : public  employee   //laborer class
{
};
```

```
int main()
   {
   manager m1, m2;
   scientist s1;
   laborer l1;

   cout << endl;            //get data for several employees
   cout << "\nEnter data for manager 1";
   m1.getdata();

   cout << "\nEnter data for manager 2";
   m2.getdata();

   cout << "\nEnter data for scientist 1";
   s1.getdata();

   cout << "\nEnter data for laborer 1";
   l1.getdata();
```

```
                                //display data for several employees
cout << "\nData on manager 1";
m1.putdata();

cout << "\nData on manager 2";
m2.putdata();

cout << "\nData on scientist 1";
s1.putdata();

cout << "\nData on laborer 1";
l1.putdata();
cout << endl;
return 0;
}
```

Virtual Base Class



Virtual base class (Cont…)

- **Virtual base class** is used to prevent the duplication/ambiguity.
- In hybrid inheritance child class has two direct parents which themselves have a common base class.
- So, the child class inherits the grandparent via two separate paths. it is also called as indirect parent class.
- All the public and protected member of grandparent are inherited twice into child.
- We can stop this duplication by making base class **virtual**.

```
class A
{
   protected:
       int i;
};
class B: virtual public A
{
 protected:
       int j;
};
class C: public virtual A
{
 protected:
       int k;
};
```

```
class D:public B, public C
{
    int sum;
  public:
    D(int v1,int v2,int v3)
    { i=v1;j=v2;k=v3; }

    void show()
    {  sum = i + j + k;
       cout<<sum;
    }
};

int main()
{
   D ob(10,20,30);
   ob.show();
}
```

VIRTUAL BASE CLASS