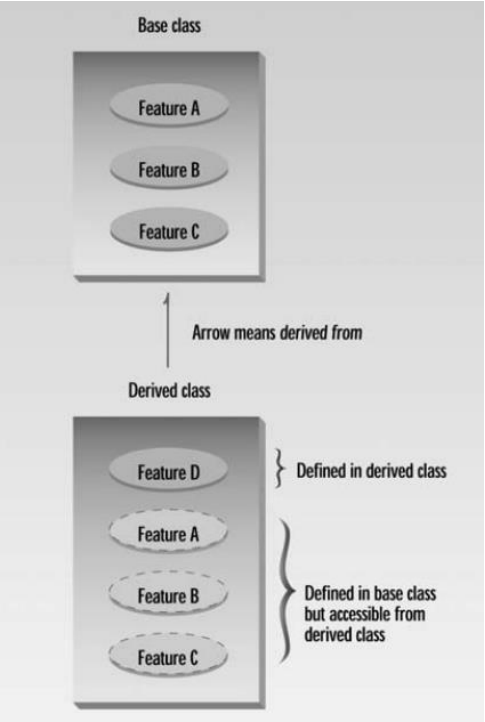
Unit 5
Inheritance

Introduction

- Inheritance is the process of creating new classes, called derived classes, from existing or base classes.
- The derived class inherits all the capabilities of the base class but can add embellishments and refinements of its own.



- Inheritance permits code reusability.
  - programmer can use a class created by another programmer and, without modifying it, derive other classes from it that are suited to particular situations.
  - Once base class is written and debugged, it can be resued without any errors and bugs.
  - Reusing existing code:
    - saves time and money and increases a program's reliability.
    - makes ease of distributing classes.

Derived class and base class

- Inheritance is a technique of building new classes from the existing classes.
- The existing classes that are used to derive new classes are called base classes and new classes derived from existing classes are called derived classes.
- When a class is derived from a base class, the derived class inherits all the characteristics of base class and can add new features.
- Base class is also called ancestor, parent, or super class and derived class is called as descendent, child or subclass.
- Because of inheritance, the base class is not changed.
- The inheritance does not work in reverse order:
  - The features in derived class can not be accessed by the base class object. However the features from base class are accessible in derived class.

- A derived class is specified by defining its relationship with the base class in addition to its own details.
- The general syntax of defining a derived class is as follows:

```
class derived_class_name: [access_specifier] base_class_name
{
    //members of derived class
};
```

- The colon(:) indicates that derived_class_name class is derived from the base_class_name class.
- The access specifiers or the visibility mode is optional and, if present can be public, private or protected. By default it is private. Visibility mode, describes the accessibility status of derived features.

- Following are some possible derived class declarations:

```
1. public derivation:
    class derived_class_name: public base_class name
    {
        //members
    };
```

  - Public members of base class remain public and protected remain protected.

```
2. protected derivation:
    class derived_class_name: protected base_class name
    {
        //members
    };
```

  - Both public members and protected members of base class remain protected in derived class.

3. **private** derivation:

```
class derived_class_name: private base_class name
{
    //members
};
```

– Public and Protected members of the base class remain private in the derived class.

4. **private** derivation by **default**:

```
class derived_class_name: base_class name
{
    //members
};
```

- In any of the above four inheritance, private data members of the base class can not be accessed by any of the derived classes.

Contd..

- A derived class inherits all base class methods with the following exceptions −
  - Constructors, and destructors of the base class.
  - Overloaded operators of the base class.
  - The friend functions of the base class.

Access Specifiers in Inheritance

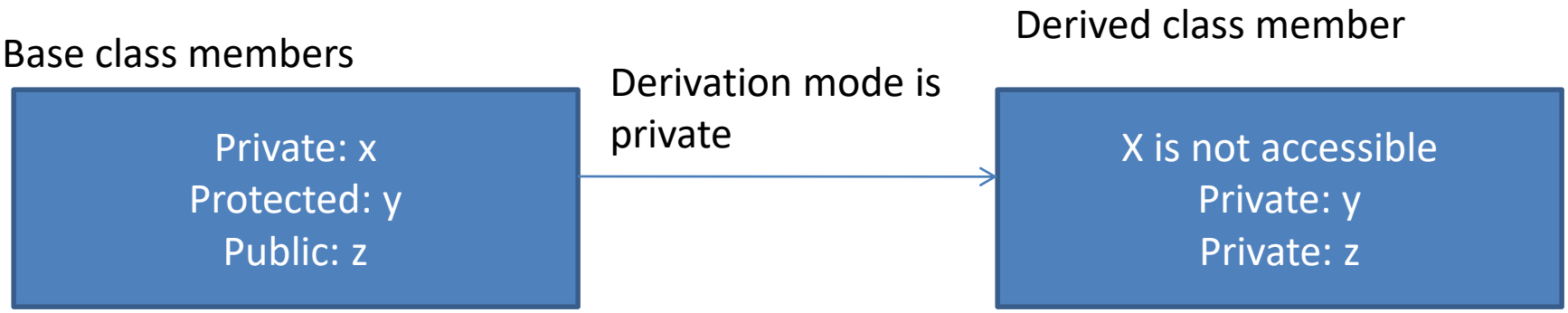
- Like the access restrictions imposed on members of class by using access specifiers we can restrict the access of inheritated members in derived class by using access specifier while performing derivation.
- Three types of access specifiers in inheritance:
  - Public mode
  - Protected mode
  - Private mode

Contd..

- **Private mode:**
  - If we derive a sub class from a base class by using a private qualifier.
  - i.e.,

```
3. private derivation:
    class derived_class_name: private base_class name
    {
        //members
    };
```

  - Then both public member and protected members of the base class will become Private in derived class.



Contd..

```
//private derivation
#include<iostream>
using namespace std;
 class base
 {
     private:
          int x;
     protected:
          int y;
     public:
          int z;
          base()
          {
               x=20;
               y=25;
               z= 30;
           }
 };
```

Contd..

```
class derive1: private base
{
    public:
        showdata()
        {
            cout<<"\n x of base class is not accessible";
            cout<<"\n y of base class ="<<y;
            cout<<"\n z of base class ="<<z;
        }
};
class derive2: private derive1
{
    public:
        displaydata()
        {
            cout<<"\n x of base class is not accessible";
            cout<<"\n y of base class is not accessible";
            cout<<"\n z of base class is not accessible";
        }
};
```

Contd..

```
int main()
{
    derive1 d1;
    derive2 d2;
    cout<<"----------For first derived class---------";
    d1.showdata();
    cout<<"\n----------For second derived class---------";
    d2.displaydata();
    return 0;
}
```

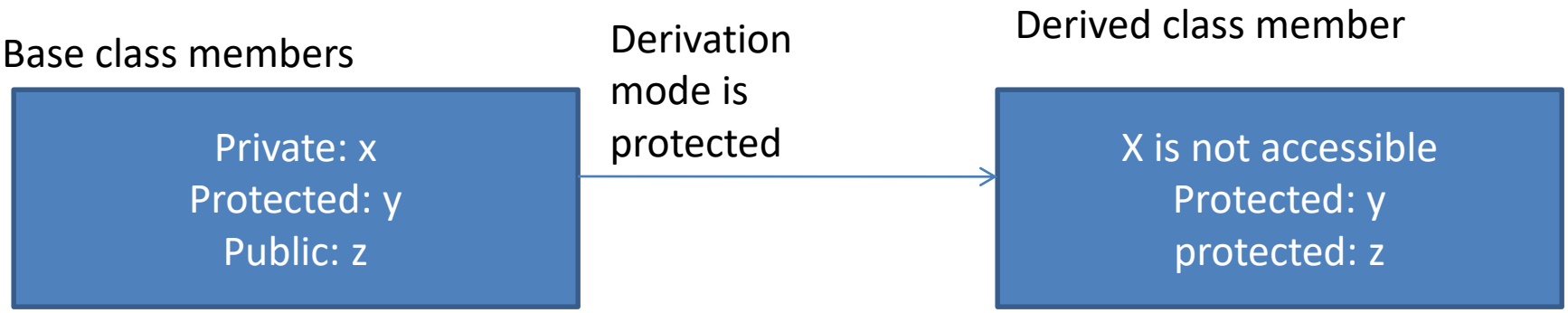
```
----------For first derived class---------
 x of base class is not accessible
 y of base class =25
 z of base class =30
----------For second derived class---------
 x of base class is not accessible
 y of base class is not accessible
 z of base class is not accessible
--------------------------------
```

Contd..

- **Protected mode**:
  - If we derive a sub class from a base class by using a protected qualifier.
  - i.e., 2. **protected** derivation:

```
class derived_class_name: protected base_class name
{
    //members
};
```

  - Then both public member and protected members of the base class will become protected in derived class.



Contd..

```
//protected derivation
#include<iostream>
using namespace std;
 class base
 {
     private:
          int x;
     protected:
          int y;
     public:
          int z;
          base()
          {
               x=20;
               y=25;
               z= 30;
           }
 };
```

Contd..

```
class derive1: protected base
{
    public:
        showdata()
        {
            cout<<"\n x of base class is not accessible";
            cout<<"\n y of base class ="<<y;
            cout<<"\n z of base class ="<<z;
         }
};
class derive2: protected derive1
{
    public:
        displaydata()
        {
            cout<<"\n x of base class is not accessible";
            cout<<"\n y of base class ="<<y;
            cout<<"\n z of base class ="<<z;
         }
};
```

Contd..

```
int main()
{
    derive1 d1;
    derive2 d2;
    cout<<"----------For first derived class---------";
    d1.showdata();
    cout<<"\n----------For second derived class---------";
    d2.displaydata();
    return 0;
}
```

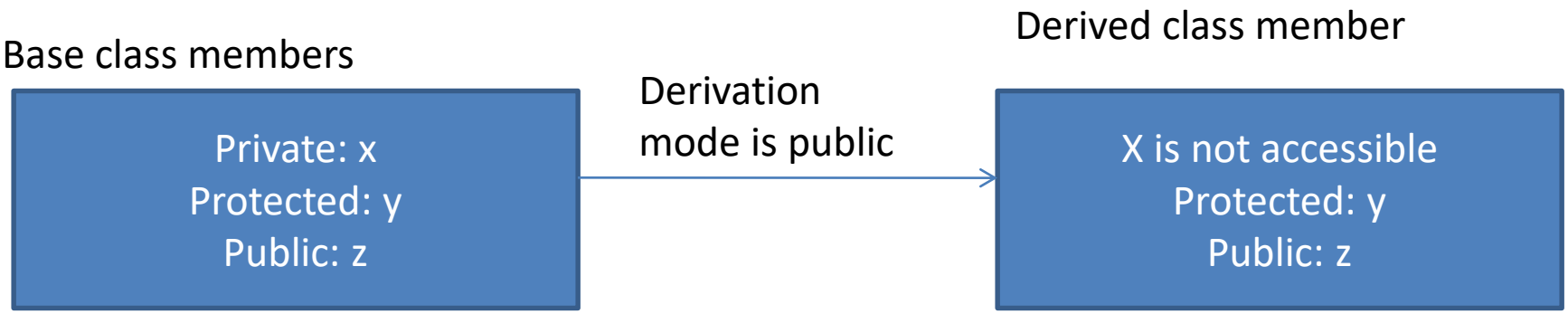
```
----------For first derived class---------
 x of base class is not accessible
 y of base class =25
 z of base class =30
----------For second derived class---------
 x of base class is not accessible
 y of base class =25
 z of base class =30
```

Contd..

- **Public mode**:
  - If we derive a sub class from a base class by using public qualifier.
    - i.e.,

```
1. public derivation:
   class derived_class_name: public base_class name
   {
       //members
   };
```

  - Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.



```
//public derivation
#include<iostream>
using namespace std;
  class base
  {
      private:
            int x;
      protected:
            int y;
      public:
            int z;
            base()
            {
                  x=20;
                  y=25;
                  z= 30;
             }
  };
```

Contd..

```
class derive1: public base
{
    public:
        showdata()
        {
            cout<<"\n x of base class is not accessible";
            cout<<"\n y of base class ="<<y;
            cout<<"\n z of base class ="<<z;
         }
};
int main()
{
    derive1 d1;
    cout<<"----------For first derived class---------";
    d1.showdata();
    return 0;
}
```

```
----------For first derived class--
x of base class is not accessible
y of base class =25
z of base class =30
```

- The below table summarizes the above three modes and shows the access specifier of the members of base class in the sub class when derived in public, protected and private modes:

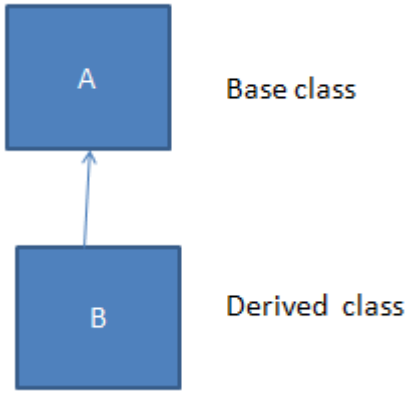
When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

Type(forms) of inheritance

- A class can inherit properties from one or more classes and from one or more levels.
- Based on the number of base classes and number of levels involved in the inheritance can be categorized into the following forms:
  - Single inheritance
  - Multiple inheritance
  - Hierarchical inheritance
  - Multilevel inheritance
  - Hybrid inheritance

Contd..

- **Single inheritance**: In this types of inheritance a derived class has only one base class.

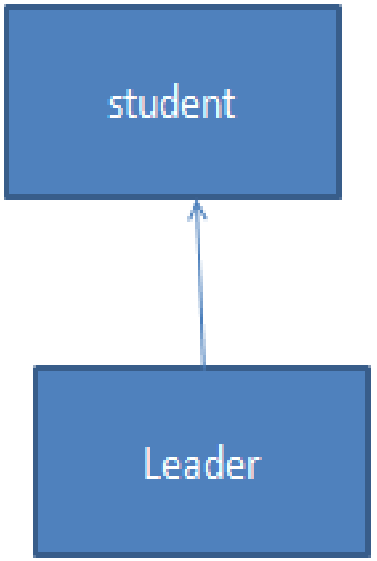


- Implementation skeleton:

```
class A
{
    //members of A
};
class B:public A
{
    //members of B
};
```

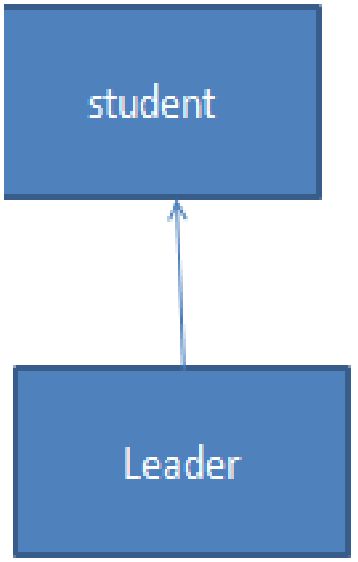
Contd..

```
//an example of single inheritance
#include<iostream>
using namespace std;
class student
{
    private:
     char name[25];
     int sid;
    public:
        void getdata()
        {
            cout<<"\n Enter Name:";
            cin>>name;
            cout<<"\n Enter student ID:";
            cin>>sid;
        }
        void showdata()
        {
            cout<<"\n Name:"<<name;
            cout<<"\n student ID:"<<sid;
        }
};
```



Contd..

```
class leader: public student
{
    private:
        char union_name[25];
    public:
        void readdata()
        {
            getdata();
            cout<<"Enter the name of associated student union:";
            cin>>union_name;
        }
        void displaydata()
        {
            showdata();
            cout<<"\n Name of associated student union:"<<union_name;
        }
};
int main()
{
    leader led;
    cout<<"Enter data on leader of student union"<<endl;
    led.readdata();
    cout<<"\n data on leader of student union:"<<endl;
    led.displaydata();
    return 0;
}
```



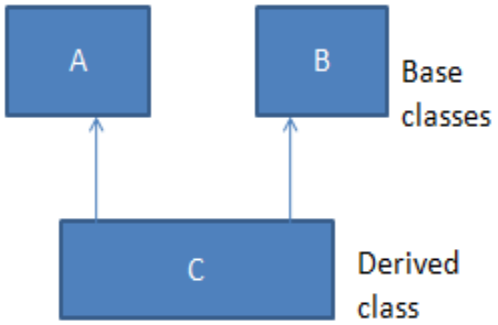
Contd..

- **Output:**

```
Enter data on leader of student union

 Enter Name:Ram

 Enter student ID:12
Enter the name of associated student union:ABC

 data on leader of student union:

 Name:Ram
 student ID:12
 Name of associated student union:ABC
```

Contd..

- **Multiple inheritance**: In this type of inheritance a single derived class may inherit from two or more than two base classes.

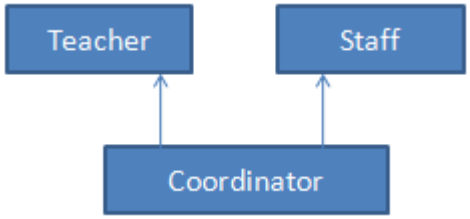


- Implementation skeleton:

```
class A
{
    // members of A
};
class B
{
    //members of B
};
class C: public A, public B
{
    //members of C
};
```

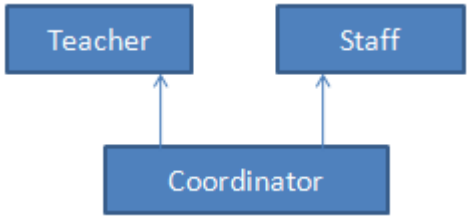
Contd..

```
//Multiple inheritance
#include<iostream>
using namespace std;
class teacher
{
    private:
        int tid;
        char subject[25];
    public:
        void getTeacher()
        {
            cout<<"Enter teacher id and subject"<<endl;
            cin>>tid>>subject;
        }
        void displayTeacher()
        {
            cout<<"Teacher ID:"<<tid<<endl;
            cout<<"Subject:"<<subject<<endl;
        }
};
```



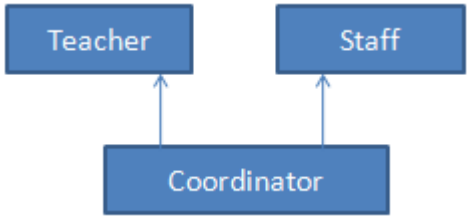
Contd..

```
class staff
{
    private:
        int sid;
        char level[25];
    public:
        void getStaff()
        {
            cout<<"Enter staff id and level"<<endl;
            cin>>sid>>level;
        }
        void displayStaff()
        {
            cout<<"staff ID:"<<sid<<endl;
            cout<<"Level:"<<level<<endl;
        }
};
```



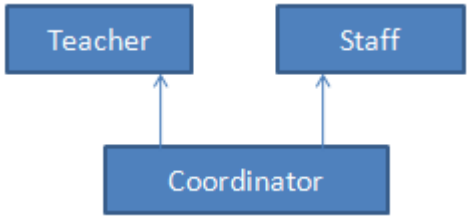
Contd..

```
class coordinator: public teacher, public staff
{
    char program[10];
    public:
        void getdata()
        {
            getTeacher();
            getStaff();
            cout<<"Enter program"<<endl;
            cin>>program;
        }
        void displaydata()
        {
            displayTeacher();
            displayStaff();
            cout<<"Program:"<<program;
        }
};
```



Contd..

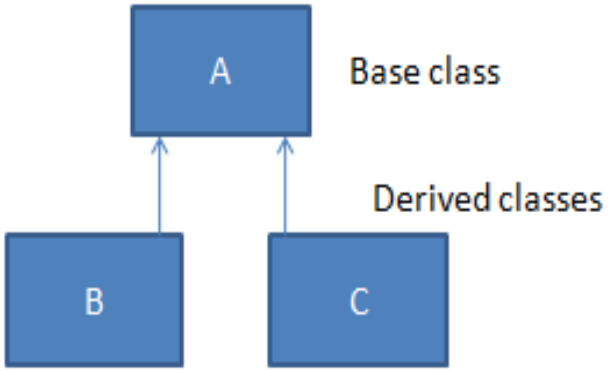
```
int main()
{
    coordinator c;
    c.getdata();
    cout<<"------------coordinator detail-------"<<endl;
    c.displaydata();
    return 0;
}
```




```
Enter teacher id and subject
10
C++
Enter staff id and level
12
8th
Enter program
CSIT
------------coordinator detail-------
Teacher ID:10
Subject:C++
staff ID:12
Level:8th
Program:CSIT
--------------------------------
```

Contd..

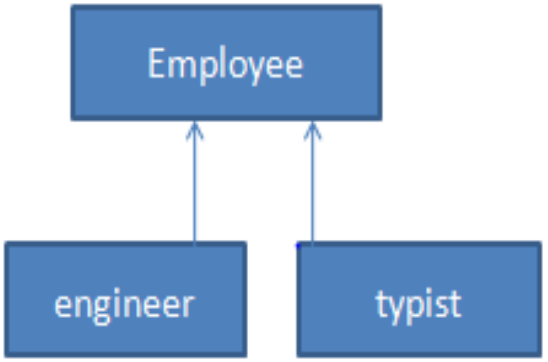
- **Hierarchical inheritance**: In this type, two or more classes inherit the properties of one base class.



- **Implementation skeleton:**

```
class A
{
    //members of A
};
class B: public A
{
    //members of B
};
class C: public A
{
    //members of C
};
```

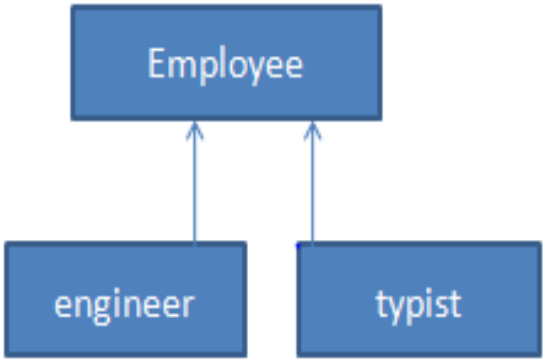
Contd..



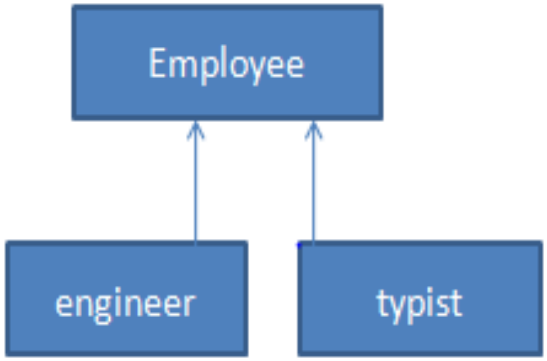

```
//Hierarchical inheritance
#include<iostream>
using namespace std;
class employee
{
    private:
        int eid, salary;
    public:
        void getEmp()
        {
            cout<<"Enter id and  salary of employee"<<endl;
            cin>>eid>>salary;
        }
        void displayEmp()
        {
            cout<<"Emp ID:"<<eid<<endl;
            cout<<"Salary:"<<salary<<endl;
        }
};
```

Contd..

```
class engineer: public employee
{
    private:
        char dept[10];
    public:
        void getdata()
        {
            getEmp();
            cout<<"Enter Department"<<endl;
            cin>>dept;
        }
        void display()
        {
            displayEmp();
            cout<<"Department:"<<dept;
        }
};
```

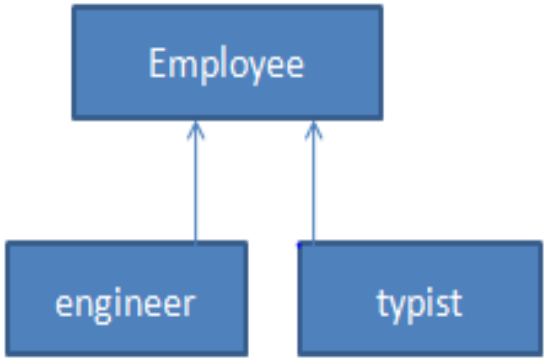


Contd..




```
class typist: public employee
{
    private:
        int ts;      //Typing speed
    public:
        void getdata()
        {
            getEmp();
            cout<<"Enter typing speed:"<<endl;
            cin>>ts;
        }
        void display()
        {
            displayEmp();
            cout<<"Typing speed:"<<ts<<endl;
        }
};
```

Contd..

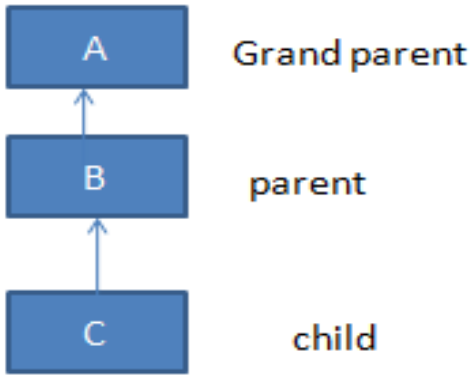



```
int main()
{
    engineer e;
    typist t;
    e.getdata();
    t.getdata();
    cout<<"--------------------Employee details---------"<<endl;
    e.display();
    cout<<endl;
    t.display();
    return 0;
}
```

```
Enter id and  salary of employee
10
45000
Enter Department
Construction
Enter id and  salary of employee
15
25000
Enter typing speed:
45
--------------------Employee details---------
Emp ID:10
Salary:45000
Department:Construction
Emp ID:15
Salary:25000
Typing speed:45
```

Contd..

- **Multilevel Inheritance**: The mechanism of deriving a class from another derived class is known as multilevel inheritance. The process can be extended to an arbitrary number of levels.



- **Implementation skeleton**:

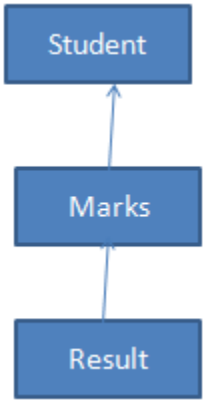
```
class A
{
    //members of A
};
class B: public A
{
    //members of B
};
class C: public B
{
    //members of C
};
```

Contd..

```
//Multilevel inheritance
#include<iostream>
using namespace std;
class student
{
    int roll;
    char name[20];
    public:
        void getStudent()
        {
            cout<<"Enter roll number and name of student:"<<endl;
            cin>>roll>>name;
        }
        void displayStudent()
        {
            cout<<"Roll Number:"<<roll<<endl;
            cout<<"Name:"<<name<<endl;
        }
};
```

Contd..

```
class marks: public student
{
    int sub1, sub2, sub3;
    public:
        void getMarks()
        {
            cout<<"Enter marks in three subjects:"<<endl;
            cin>>sub1>>sub2>>sub3;
        }
        void displayMarks()
        {
            cout<<"Subject1:"<<sub1<<endl;
            cout<<"Subject2:"<<sub2<<endl;
            cout<<"Subject3:"<<sub3<<endl;
        }
        int findTotalMarks()
        {
            return sub1+sub2+sub3;
        }
};
```



Contd..

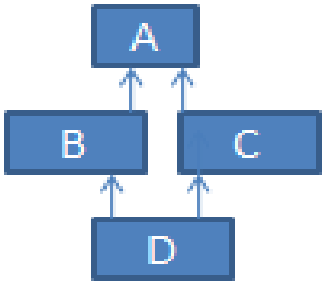
```
class result: public marks
{
    float total, percentage;
    public:
        void getdata()
        {
            getStudent();
            getMarks();
        }
        void displaydata()
        {
            displayStudent();
            displayMarks();
            total=findTotalMarks();
            percentage=total/3;
            cout<<"Total marks:"<<total<<endl;
            cout<<"Percentage:"<<percentage;
        }
};
int main()
{
    result r;
    r.getdata();
    cout<<"----------Result Details---------"<<endl;
    r.displaydata();
    return 0;
}
```

Contd..

```
Enter roll number and name of student:
10
Ram
Enter marks in three subjects:
90
89
91
----------Result Details--------
Roll Number:10
Name:Ram
Subject1:90
Subject2:89
Subject3:91
Total marks:270
Percentage:90
--------------------------------
```

Contd..

- **Hybrid Inheritance:** This type of inheritance includes more than one type of inheritance.



Grand parent

Parent

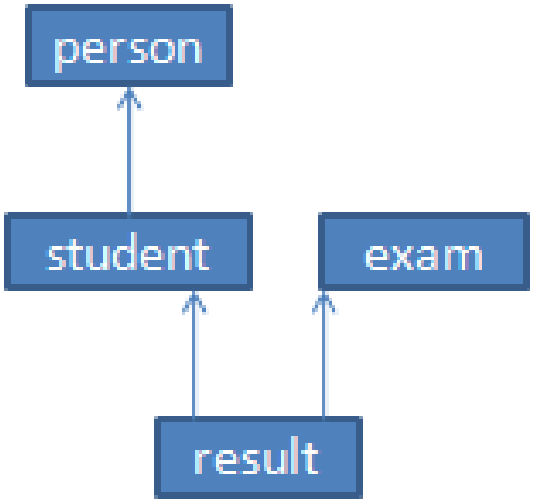
child

- **Implementation skeleton:**

```
class A
{
    //members of A
};
class B: public A
{
    //members of B
};
class C: public A
{
    //members of C
};
class D:public B, public C
{
    //members of D
};
```

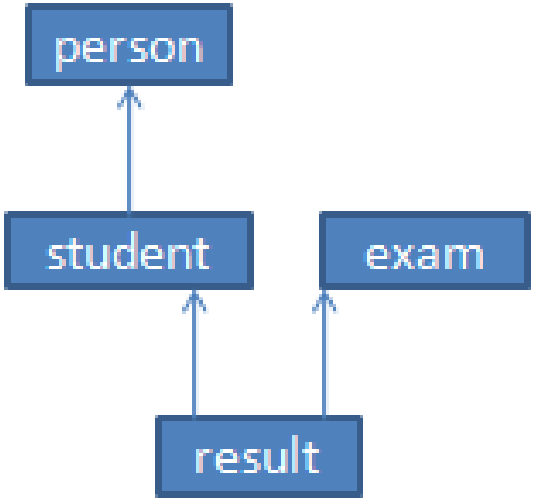
Contd..

```
//hybrid inheritance
#include<iostream>
using namespace std;
class person
{
    private:
        char name[25];
        int age;
    public:
        void getdata()
        {
            cout<<"\n Enter Name:";
            cin>>name;
            cout<<"Enter age:";
            cin>>age;
        }
        void showdata()
        {
            cout<<"\n Name:"<<name;
            cout<<"\n Age:"<<age;
        }
};
```



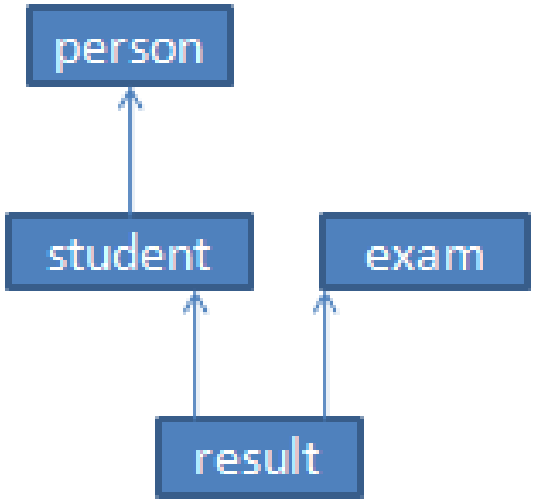
Contd..

```
class exam
{
    protected:
        int m1, m2, m3;
    public:
        void getdata()
        {
            cout<<"Enter marks in three subjects:";
            cin>>m1>>m2>>m3;
        }
        void showdata()
        {
            cout<<"\n marks in subject1:"<<m1;
            cout<<"\n marks in subject2:"<<m2;
            cout<<"\n marks in subject3:"<<m3;
        }
};
```



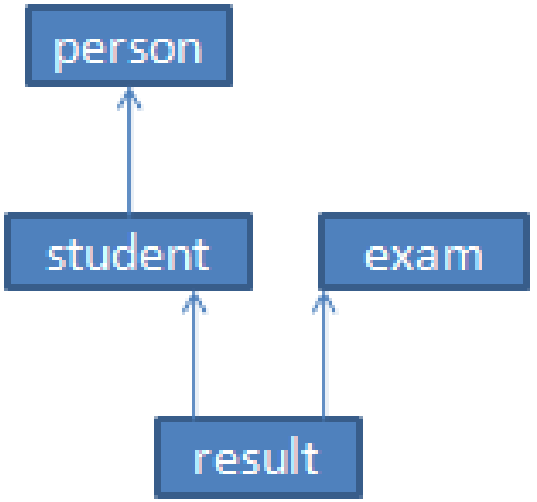
Contd..

```
class student:public person
{
    private:
        int sId;
        char stream[10];
    public:
        void getdata()
        {
            person::getdata();
            cout<<"\n Enter student ID:";
            cin>>sId;
            cout<<"\n Enter stream:";
            cin>>stream;
        }
        void showdata()
        {
            person::showdata();
            cout<<"\n Student Id:"<<sId;
            cout<<"\n Stream:"<<stream;
        }
};
```



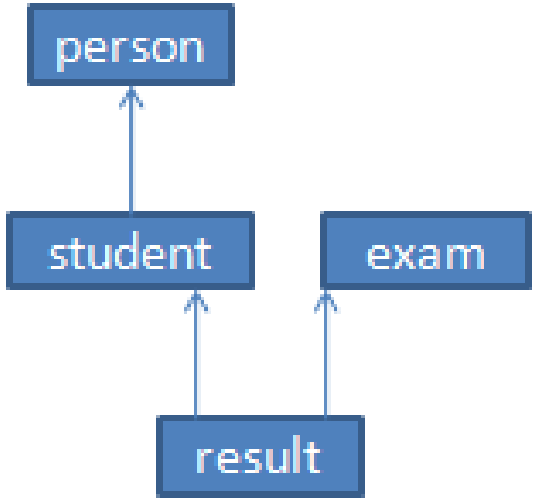
Contd..

```
class result:public student, public exam
{
    private:
        int total;
    public:
        void getdata()
        {
            student::getdata();
            exam::getdata();
        }
        void showdata()
        {
            student::showdata();
            exam::showdata();
            total = m1+m2+m3;
            cout<<"\n Total marks:"<<total;
            cout<<"\n percentage:"<< float(total)/3;
        }
};
```



Contd..

```
int main()
{
    result r;
    cout<<"Enter data for result processing:"<<endl;
    r.getdata();

    cout<<"\n data on student:"<<endl;

    r.showdata();
    return 0;
}
```



Contd..

- **Output:**

```
Enter data for result processing:

 Enter Name:Ram
Enter age:20

 Enter student ID:1001

 Enter stream:science
Enter marks in three subjects:
90
91
89

 data on student:

 Name:Ram
 Age:20
 Student Id:1001
 Stream:science
 marks in subject1:90
 marks in subject2:91
 marks in subject3:89
 Total marks:270
 percentage:90
--------------------------------
```

Overriding member functions

- We can define the function in derived class having same name and signature as that of base class which is called function overriding.
- It is called overriding because the new name overrides(hides or displaces) the old name inherited from base class.
- In such situations derived class have two versions of same function one derived from base class and another defined in the derived class itself.
- And if we call the overridden function by using the object of derived class version of the method defined in the derived class is invoked.
- We can call the version of method derived from base class as follows:

```
obj.class_name::method_name
```

Contd..

```
//method overriding
#include<iostream>
using namespace std;
class A
{
    public:
        void show()
        {
            cout<<"This is class A";
        }
};
class B: public A
{
    public:
        void show()
        {
            cout<<"This is class B"<<endl;
        }
};
int main()
{
    B b;
    b.show();//Invoking the member function from class B
    b.A::show();//Invoking the member function from class A
    return 0;
}
```

```
This is class B
This is class A
```

Constructors in derived classes

- Constructors play an important role in initializing objects.
- As long as base class constructor takes no any arguments, the derived class need not have a constructor function.
- However, if any base class contains a constructor with one or more arguments , then it is mandatory for the derived class to pass arguments to the base class constructor.
- Initial values are supplied to all classes when a derived class object is declared.
- 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.
- So, When object of a derived class is created, the constructor of the base class is executed first and then the constructor of the derived class is executed next.

Contd..

```
//derived class constructor
    #include<iostream>
    using namespace std;
    class A
    {
        protected:
            int adata;
        public:
            A(int a)
            {
                adata =a;
            }
    };
    class B: public A
    {
        private:
            int bdata;
        public:
            B(int a, int b): A(a)
            {
                bdata=b;
            }
            void showdata()
            {
                cout<<"adata="<<adata<<endl;
                cout<<"bdata="<<bdata<<endl;
            }
    };
```

```
int main()
{
    B b(10, 100);
    b.showdata();
    return 0;
}
```

```
adata=10
bdata=100
```

Contd..

- If the base class contains no constructor, we can write the derived class constructor as follows:

```
B(int a, int b)
{
     adata= a;
     bdata=b;
}
```

Contd..

- In case of multiple inheritance, constructor in the base classes are placed in the initializer list in the derived class constructor separated by commas.

- The order of the constructor invocation is not as the order in the initializer list but as the order of the inherited base class mentioned in derived class declaration.

```
//derived class constructor
    #include<iostream>
    using namespace std;
    class A
    {
        protected:
            int adata;
        public:
            A(int a)
            {
                adata =a;
            }
    };
    class B
    {
        private:
            int bdata;
        public:
            B( int b)
            {
                bdata=b;
            }
    };
    class C: public A, public B
    {
        private:
            int cdata:
        public:
            C(int a, int b, int c): A(a),
            {
                cdata=c;
            }
    };
```

Contd..

- Order of execution of constructors:
  - The base class constructor is executed first and then constructor in the derived class is executed.
  - In case of multiple inheritance, the base class constructors are executed in which they appear in the definition of the derived class.
  - similarly, in a multilevel inheritance, the constructors will be executed in the order of inheritance. i.e., the constructor of the root base class is called at first and the constructor of the lastly derived class is called sequentially at last.
  - Furthermore, the constructors for virtual base classes are invoked before any non-virtual base classes.
  - If there are multiple virtual base classes, they are invoked in the order in which they are declared in the derived class.

Contd..

```
//order of execution of constructors
#include<iostream>
using namespace std;
class A
{
    public:
        A()
        {
            cout<<"Class A constructor"<<endl;
        }
};
class B:public A
{
    public:
        B( )
        {
            cout<<"Class B constructor"<<endl;
        }
};
class C:  public B
{
    public:
        C()
        {
            cout<<"Class C constructor"<<endl;
        }
};
int main()
{
    C x;
    return 0;
}
```

```
Class A constructor
Class B constructor
Class C constructor
```

Destructors in derived classes

- Destructor is a member function which destructs or deletes an object.
- Order of destructor invocation is just reverse order of constructor invocation.

Contd..

- Order of execution of destructors:
  - The derived class destructor is executed first and then the destructor in the base class is executed.
  - In case of multiple inheritance, the destructor of derived class is called first and the destructor of the base class which is mentioned at last in the derived class declaration is called next and destructor of other classes from last towards first is called sequentially and the destructor of the class which is mentioned at first is called last..
  - similarly, in a multilevel inheritance, the destructors will be executed in the reverse order of inheritance.

Abstract Base Class

- Sometimes the logic cannot be defined for all the functions present in the base class since the implementation is not known in advance.
- But if we know that each derived class must mandatorily implement that particular function whose logic is unknown in advance then such functions can be implemented as abstract functions.
- Functions which are declared only but contain no any logic are called abstract functions or pure virtual functions.
- Class having at least one abstract function is called abstract class.
- If an abstract function is defined in base class, then any derived class that inherits the base class must provide its own implementation for the function compulsorily

```cpp
#include <iostream>
using namespace std;
class shape
{  public:
    int l,b;
    void setdata(int la, int bb)
    {   l =la;
        b =bb;
    }
    virtual void calculate() = 0;
};
class rect: public shape
{   public:
    void calculate(){
        cout<<"The Area is "<< l * b <<endl;
    }
};
```

```cpp
class square: public shape
{
    public:
    void calculate(){
        cout<<"The area is "<< l*l<<endl;
    }

};

int main() {
    rect r1;
    r1.setdata(4,5);
    r1.calculate();
    square s1;
    s1. setdata(6,6);
    s1.calculate();
    return 0;
}
```

```
The Area is 20
The area is 36
```

Example:

```
//destructor under inheritance
#include<iostream>
using namespace std;
class A
{
    public:
        ~A()
        {
            cout<<"Class A destructor"<<endl;
        }
};
    class B:public A
{
    public:
        ~B()
        {
            cout<<"Class B destructor"<<endl;
        }
};
    class C:public B
{
    public:
        ~C()
        {
            cout<<"Class C destructor"<<endl;
        }
};
int main()
{
    {
        C x;
    }//destructor is called at this point
    return 0;
}
```

```
Class C destructor
Class B destructor
Class A destructor
```

Aggregation(Containership)

- Two types of class relationships:
  - Aggregation and
  - Inheritance

Contd..

- **Inheritance:**
  - Inheritance is often called a "kind-of" or "is-a" relationship. In inheritance If a class B is derived by inheritance from a class A, we can say that "B is a kind of A." This is because B has all the characteristics of A, and in addition some of its own.
  - E.g., we can say that bulldog is a kind of dog: A bulldog has the characteristics shared by all dogs but has some distinctive characteristics of its own.

Contd..

- **Aggregation:**
  - This is another type of relationship, called a "has a" relationship, or containership.
    - We say a library has a book .
  - Aggregation is also called a "part-whole" relationship: the book is part of the library.
- In object-oriented programming, aggregation(has a relationship) may occur when one object is an attribute of another. i.e., when one object is contained in another.

```
class A
   {
   };
class B
   {
   A objA;  // define objA as an object of class A
   };
```

Contd… Example

```
//containership
#include<iostream>
using namespace std;
class employee
{
    private:
        int eid,salary;
    public:
        void getdata()
        {
            cout<<"Enter id and salary of employee:"<<endl;
            cin>>eid>>salary;
        }
        void display()
        {
            cout<<"Employee Id:"<<eid<<endl;
            cout<<"Employee salary:"<<salary<<endl;
        }
};
```

Contd..

```
class company
{
    private:
        int cid;
        char cname[20];
        employee e; // containership(object of employee class is included in company class
    public:
        void getdata()
        {
            cout<<"Enter id and name of the company:"<<endl;
            cin>>cid>>cname;
            e.getdata();
        }
        void display()
        {
            cout<<"Company Id:"<<cid<<endl;
            cout<<"company Name:"<<cname<<endl;
            e.display();
        }
};
```

Contd..

```
int main()
{
    company c;
    c.getdata();
    cout<<"-------Company Details-------"<<endl;
    c.display();
    return 0;
}
```

- **OUTPUT:**

```
Enter id and name of the company:
1001
CAB
Enter id and salary of employee:
2001
35000
-------Company Details-------
Company Id:1001
company Name:CAB
Employee Id:2001
Employee salary:35000

--------------------------------
```

Ambiguity in Inheritance

- **Ambiguity in multiple inheritance:**
  - Suppose two base classes have an exactly similar member.
  - Also, suppose a class derived from both of these classes has not this member.
  - Then, if we try to access this member from the objects of the derived class, it will be ambiguous.
  - We can remove this ambiguity by using the syntax:

    Obj.class_name::methodname.

Contd..

```
//Ambiguity in multiple inheritance and removal of ambiguity
    #include<iostream>
    using namespace std;
    class A
    {
        public:
            void show()
            {
                cout<<"This is class A"<<endl;
            }
    };
    class B
    {
        public:
            void show()
            {
                    cout<<"This is class B"<<endl;
            }
    };
```

Contd..

```
class C: public A, public B
{

};
int main()
{
    C c;
//  c.show();    // ambiguous and will not compile
    c.A::show(); // OK
    c.B::show();  //ok
    return 0;
}
```

Output

```
This is class A
This is class B

--------------------------------
```

Error will be:

Line	Col	File	Message
		C:\Users\hp\Documents\C++ practice\Inheritance\a...	In function 'int main()':
27	4	C:\Users\hp\Documents\C++ practice\Inheritance\ambi...	[Error] request for member 'show' is ambiguous
15	9	C:\Users\hp\Documents\C++ practice\Inheritance\ambi...	[Note] candidates are: void B::show()
7	9	C:\Users\hp\Documents\C++ practice\Inheritance\ambi...	[Note] void A::show()

Contd..

- We can also remove this ambiguity by adding a function in class C as follows:

```
class C: public A, public B
{
    public:
        void show()
        {
            A::show();
            B::show();
        }
};
```

Homework

- Differentiate between base class and derived class with suitable examples.
- Differentiate between private, public and protected variables with suitable example.
- Explain the role of inheritance in OOP. What is public, private and protected dentations? Explain.
- Discuss a situation in which the private derivation will be more appropriate as compared to public derivation.
- What is inheritance? Explain the types of inheritance with suitable examples.
- What is multilevel inheritance? How it differ from multiple inheritance?
- Define a shape class (with necessary constructors and member functions) in Object Oriented Programming (abstract necessary attributes and their types). Write a complete code in C++ programming language.
  - Derive triangle and rectangle classes from shape class adding necessary attributes.
  - Use these classes in main function and display the area of triangle and rectangle.

Contd..

- Define a **student** class (with necessary constructors and member functions) in Object Oriented Programming (abstract necessary attributes and their types). (Write a complete code in C++ programming language).
    - Derive a **Computer Science and Mathematics** class from **student** class adding necessary attributes (at least three subjects).
    - Use these classes in a main function and display the averages marks of computer science and mathematics students.

- Differentiate between function overriding and function overloading. Explain with suitable example.

- Define a **clock** class (with necessary constructors and member functions) in Object Oriented Programming (abstract necessary attributes and their types). (Write a complete code in C++ programming language).
    - Derive a **wall_clock** class from **clock** class adding necessary attributes.
    - Create two objects of **wall_clock** class with all initial state to 0 or NULL.

Contd..

- What is container class? Differentiate container class from inheritance.
- Differentiate between overloading and overriding.
- Differentiate abstract base class and concrete classes with suitable example.
- What are ambiguities in inheritance and how do you resolve those ambiguities.
- What is the purpose of virtual base classes? Explain with suitable example.

End of Unit 5