Inheritance

Inheritance

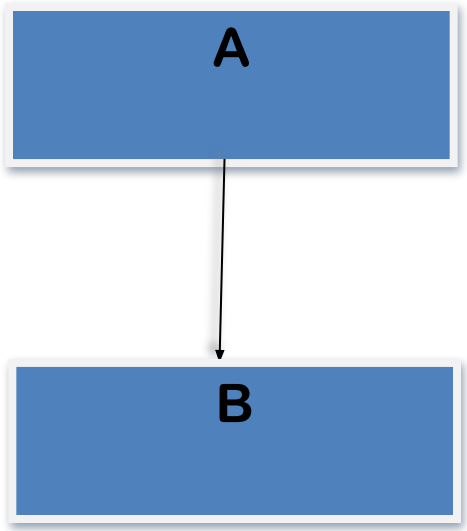
- **Inheritance/ derivation** is a mechanism of deriving a new class from an exiting class.
- The existing/ old class is referred to as the *base class* and the new one is called the *derived class* or *subclass*.
- Inheritance supports the concept of reusability.

Types of inheritance

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance

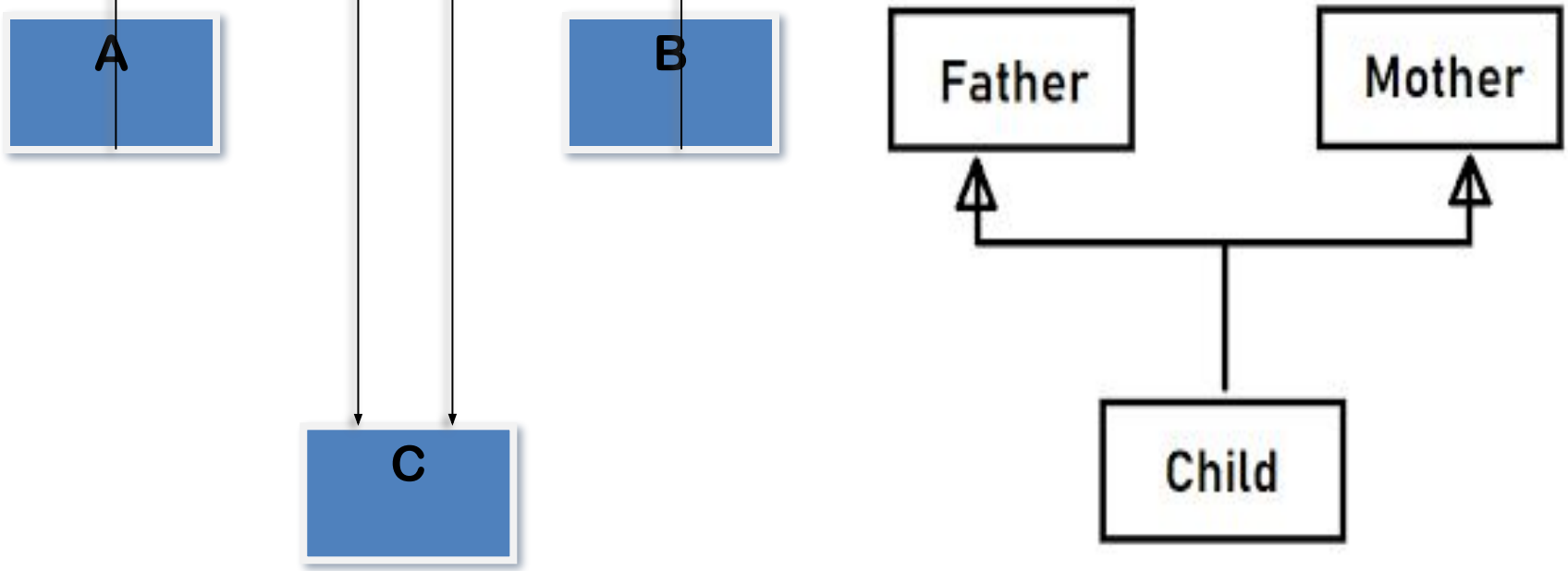
Single Inheritance

- A derived class with only one base class is called single inheritance



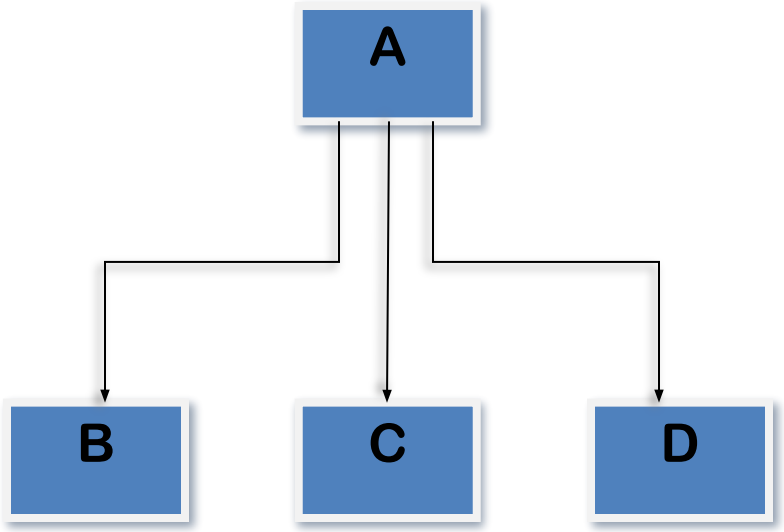
Multiple Inheritance

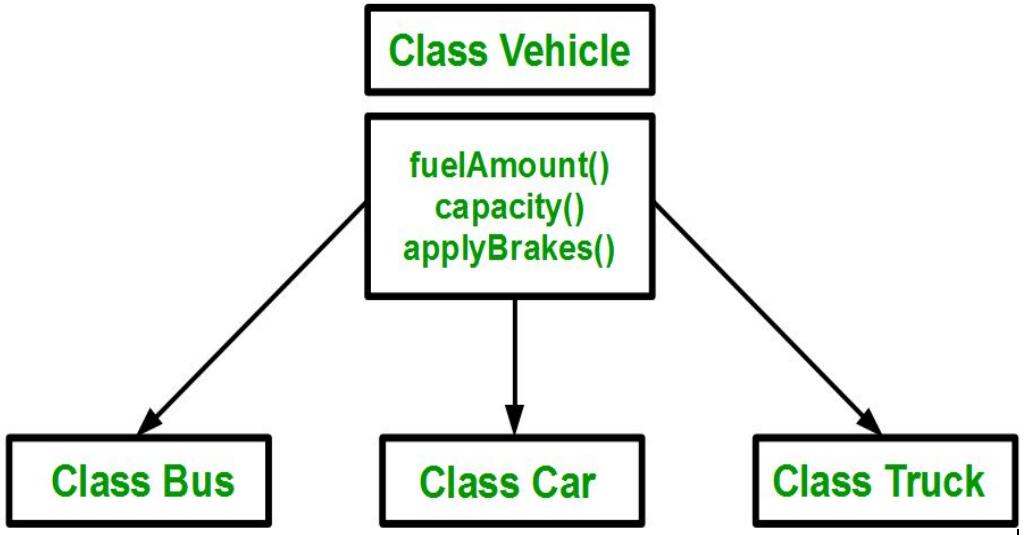
- Derived class with several base classes is called multiple inheritance.



Hierarchical Inheritance

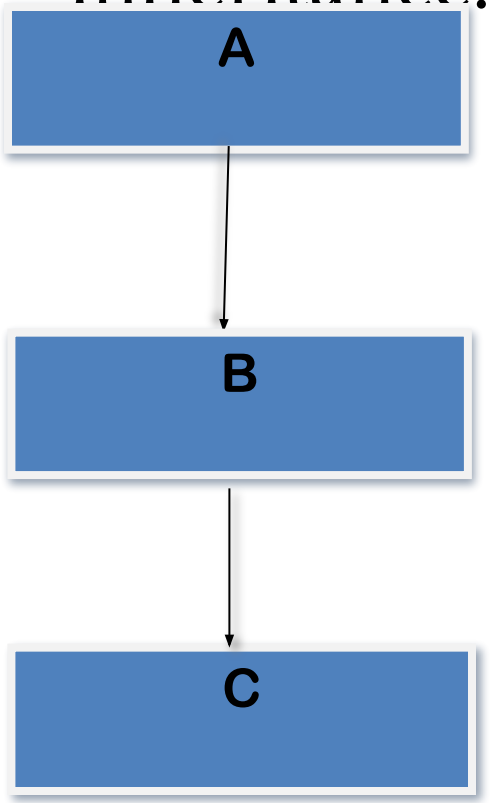
- The properties of one class can be inherited by more than one class. This process is known as hierarchical inheritance.





Multilevel Inheritance

- The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance.



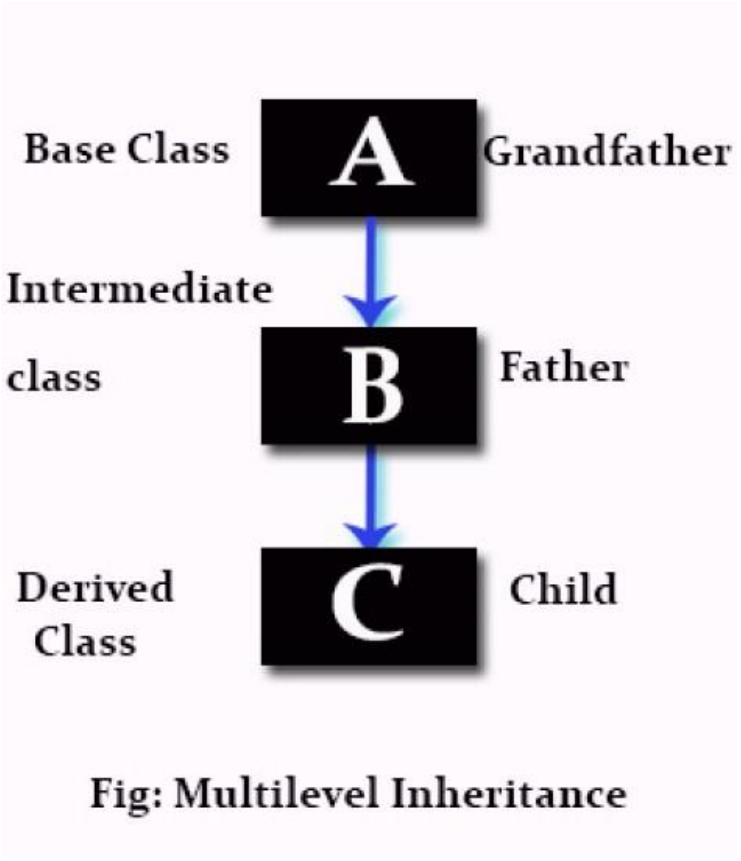
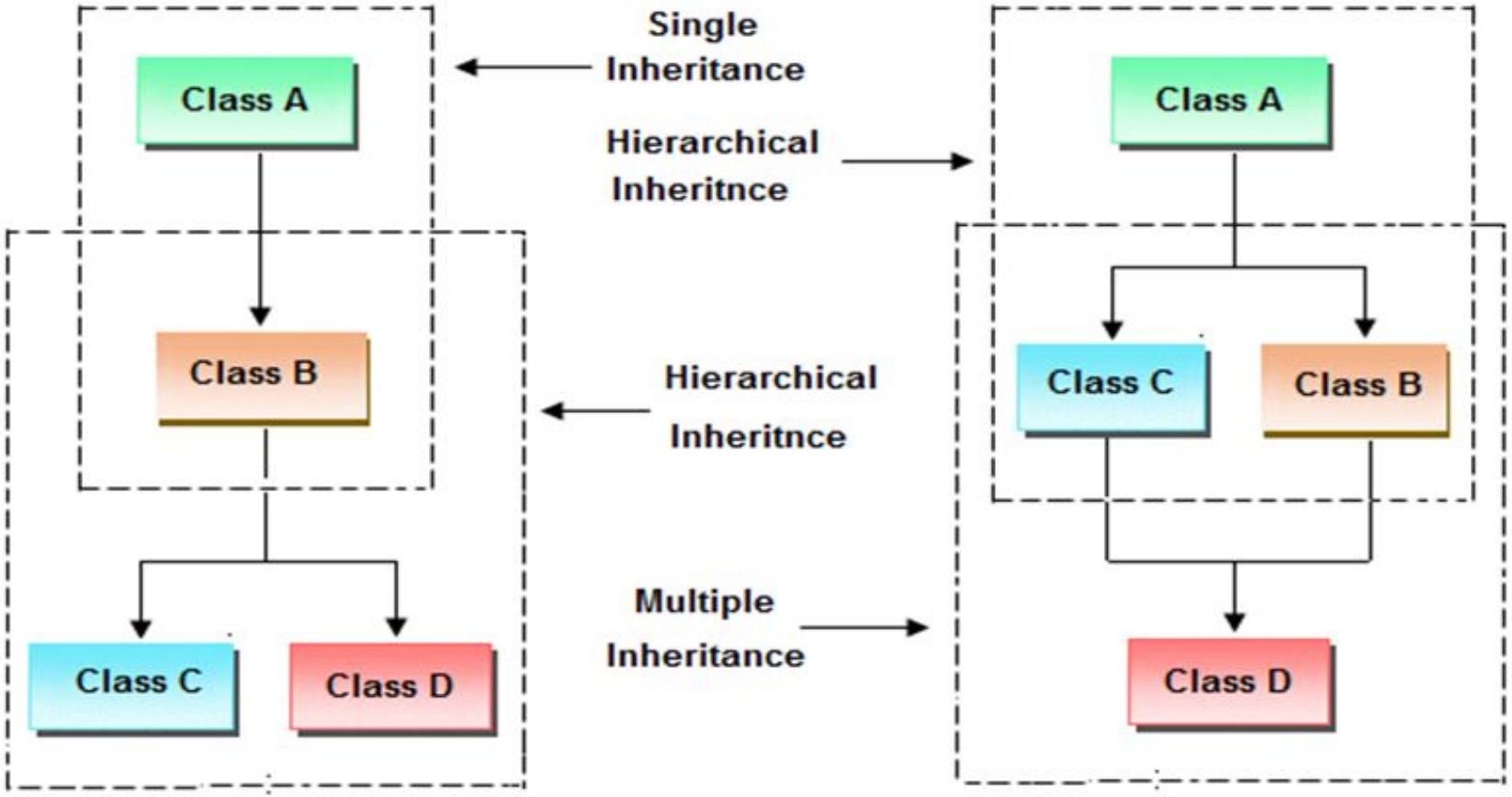


Fig: Multilevel Inheritance

Hybrid Inheritance

- Combination of any two or more inheritance types



Defining Derived classes

- Syntax:-

```
class derived_class-name : visibility-mode
   base-class-name
{
   ..............
   ..............//members of derived class
   ..............
};
```

- The **colon** indicates that the derived-class-name is derived from the base-class-name.
- The **visibility-mode** may be **private, protected** and **public.**
- The default visibility-mode is private.
- Visibility mode specifies whether the features of the base class are derived privately /publicly/ protectedly.

Examples

- class ABC : private XYZ //private derivation or private inheritance
  {
   members of ABC
  };
- class ABC : public XYZ //public derivation or public inheritance
  {
   members of ABC
  };
- class ABC : XYZ   //private derivation by default
  {
   members of ABC
  };

Important Points to remember

- Public Inheritance: When the base class is ***publicly*** inherited by a derived class, **public members** of a base class become **public members** of **derived class**; Therefore the public members of the base class can be accessed by the member function as well as objects of the derived class.
- Private Inheritance: When the base class is ***privately*** inherited by a derived class, '**public members**' of the base class become '**private members**' of **derived class**. Therefore the public members of the base class can only be accessed by the member function of the derived class.
- The private members are not inherited in terms of **access**, these are **included** in derived class that means the memory of all private data members of the base class will be included in derived class object. Therefore, the private members of base class will never become

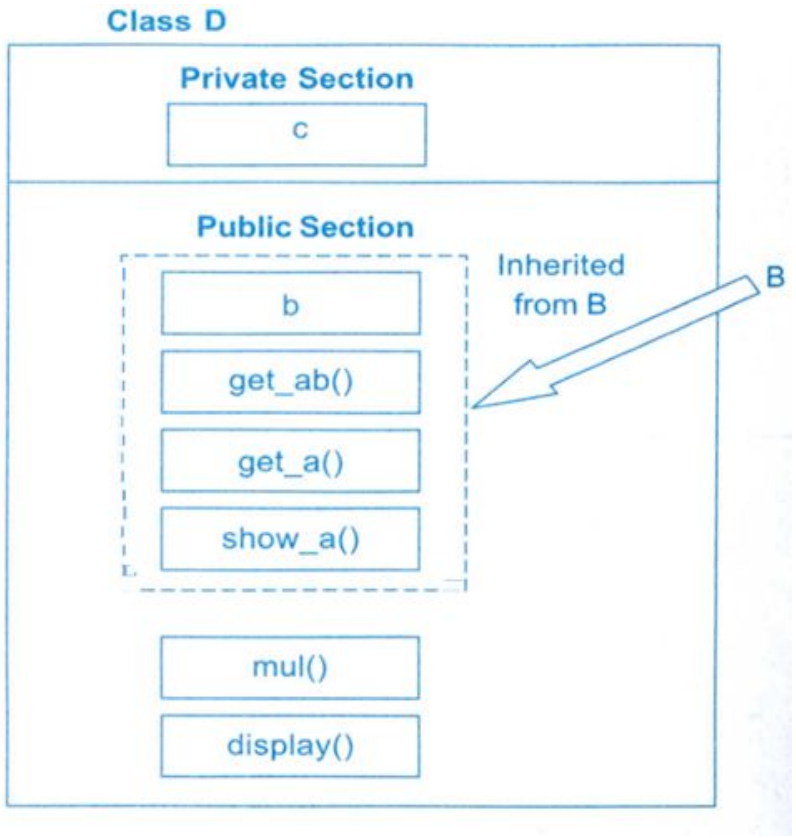
Single Inheritance(Example)

//single Inheritance: PUBLIC Inheritance

```
#include<iostream>
using namespace std;
class B
{   int a;
    public:
    int b;
    void get_ab();
    int get_a(void);
    void show_a(void);
};
```

```
class D:public B
{   int c;
    public:
    void mul(void);
    void display(void);
    };
```

How many members are there in Class D



Single Inheritance(Example) Cont..

```
void B:: get_ab(void)
{
cout<<“Enter a and
   b:";
cin>>a>>b;}
int B::get_a()
{
   return a;
}
void B::show_a()
{
   cout<<"a="<<a<<“\n";
}
```

```
void D::mul()
{
   c=b*get_a();
}
void D :: display()
{
   cout<<"a="<<get_a()<<“\
   n";
   cout<<"b="<<b<<“\n";
   cout<<"c="<<c<<"\n\n";
}
```

Single Inheritance(Example) Cont..

```
int main()
{
   D d;
   d.get_ab();
   d.mul();
   d.show_a();
   d.display();

   d.b=20;
   d.mul();
   d.display();
   return 0;
}
```

Output:
Enter a and b: 5 10

a=5
a=5
b=10
c=50

a=5
b=20
c=100

Private Inheritance

- In private derivation, the public members of the base class become private members of the derived class. Therefore, the objects of D can not have direct access to the public member functions of B.

*

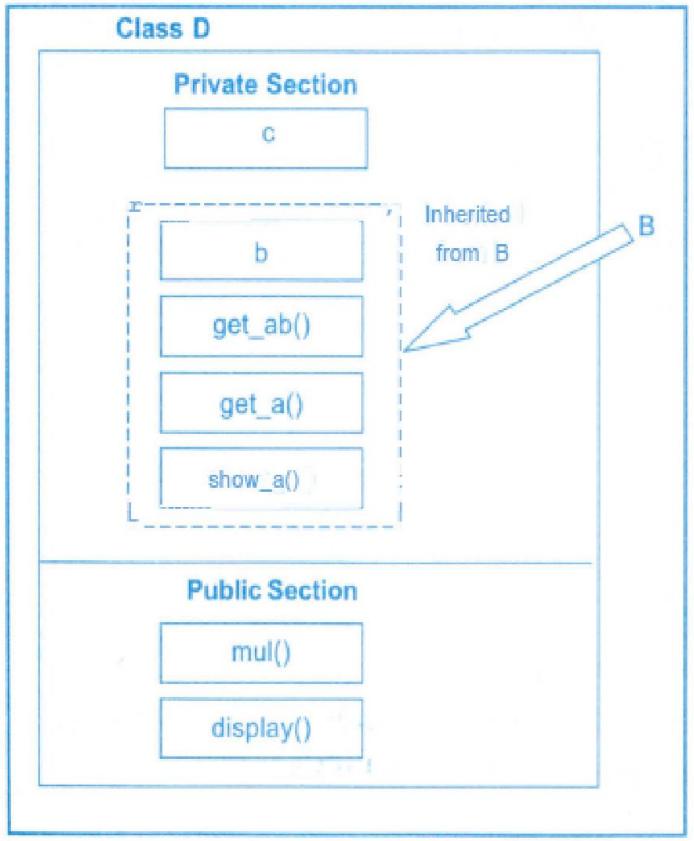
Single Inheritance(Example)

//single Inheritance: PRIVATE Inheritance

```
#include<iostream>
using namespace std;
class B
{   int a;
    public:
    int b;
    void get_ab();
    int get_a(void);
    void show_a(void);
};
```

```
class D:private B
{   int c;
    public:
    void mul(void);
    void      display(void);
    };
```

How many members are there in Class D



Single Inheritance(Example) Cont..

Public inheritance

```
void B:: get_ab(void)
{cout<<“Enter a and b:";
   cin>>a>>b; }
int B::get_a()
{   return a; }
void B::show_a()
{   cout<<"a="<<a<<“,";
}
```

```
void D::mul()
{   c=b*get_a(); }
void D :: display()
{
    cout<<"a="<<get_a()<<“,";
    cout<<"b="<<b<<“,";
    cout<<"c="<<c<<"\n";
}
```

Private inheritance

```
void B:: get_ab(void)
{ cout<<"Enter a and
    b:";
cin>>a>>b; }
int B::get_a()
{   return a; }
void B::show_a()
{   cout<<"a="<<a<<“,";
}
```

```
void D::mul()
{   get_ab();
    c=b*get_a();  }
void D :: display()
{   show_a();
    cout<<"b="<<b<<“,";
    cout<<"c="<<c<<"\n\n
    ";
}
```

Single Inheritance(Example) Cont..

Public inheritance

```
int main()
{
    D d;
    d.get_ab();
    d.mul();
    d.display();
    d.b=20;
    d.mul();
    d.display();
    return 0; }
```

```
Output:
Enter a and b:5 10
a=5 ,b=10 ,c=50
Enter a and b:5 20
a=5 ,b=20 ,c=100
```

Private inheritance

```
int  main()
{  D d;
    d.mul();
    d.display();
    d.mul();
    d.display();
    return 0;
}
```

```
Output:
Enter a and b:5 10
a=5 ,b=10 ,c=50
Enter a and b:5 20
a=5 ,b=20 ,c=100
```

What will be the Output?

```
#include<iostream>
using namespace std;
class B
{   int a;
    public:   B(){   cout<<"\nContructor B() is called\n“;}
};
class D:public B
{   int c;
    public:   D(){   cout<<"\nContructor D() is called\n“;}
};
int main()
{
    D d;
    B b;
    return 0;
}
```

Output:
Contructor B() is called
Contructor D() is called
Contructor B() is called

*

Inheritable

How to inherit private data of base class in derived class?

Solution1: by changing access specifier of members as private to public. **This violates the data hiding(not a good solution).**

Solution2: By using **Protected access specifier** : A member declared as protected is accessible by the member functions within its class and any class immediately derived from it. It cannot be accessed by the functions outside these two classes. A class can now use all the three visibility modes as illustrated below:

```
class alpha
{
   private:       //visible to member functions within its class
      ……..
      ……..
   protected :   //visible to member functions within its own
   class and          …….          immediately derived class
      ……..
   public:             //visible to all functions in the class
      ……..
```

Cont...

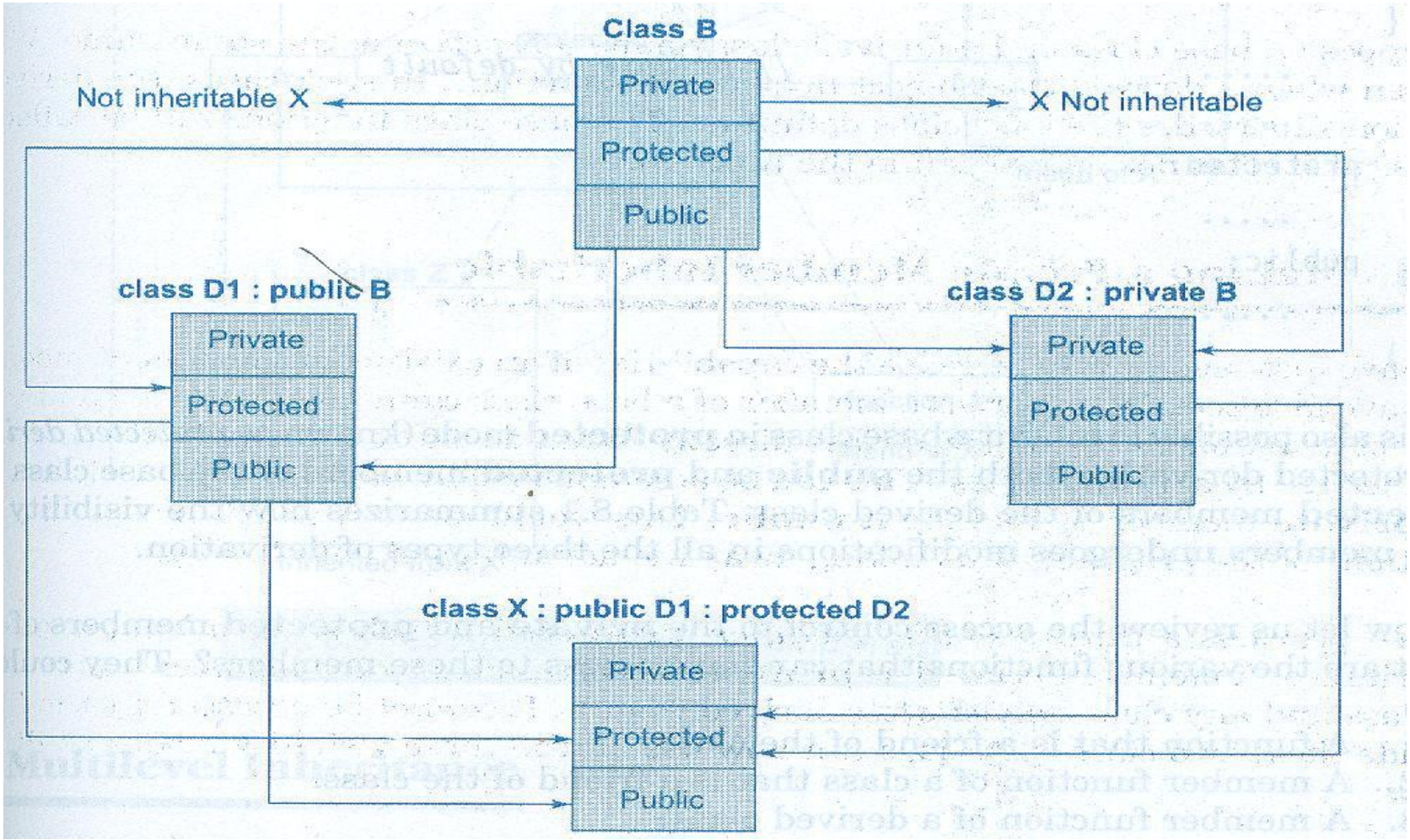
The keyword private, protected, and public may appear in any order and any number of times in the declaration of a class. For example,

```
class B
{
   protected:
      ……..
      ……..
   protected :
      ……..
      ……..
   public:
      ……..
      ……..
   };
```

Important Points to remember

- **Public Inheritance**: When the base class is ***publicly*** inherited by a derived class protected members of base class become protected members of derived class. Therefore the protected members of the base class can be accessed by only member functions of the derived class . **It is also available for further inheritance.**
- **Private Inheritance:** When the base class is ***privately*** inherited by a derived class, protected members of base class become private members of derived class. Therefore the protected members of the base class can only be accessed by the member function of the derived class. **No further inheritance allowed.**

Effect of inheritance on visibility of members



Visibility of inherited members

Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private --Z	Not inherited	Not inherited	Not inherited
Protected ---?	Protected	Private	Protected
Public ---?	Public	Private	Protected

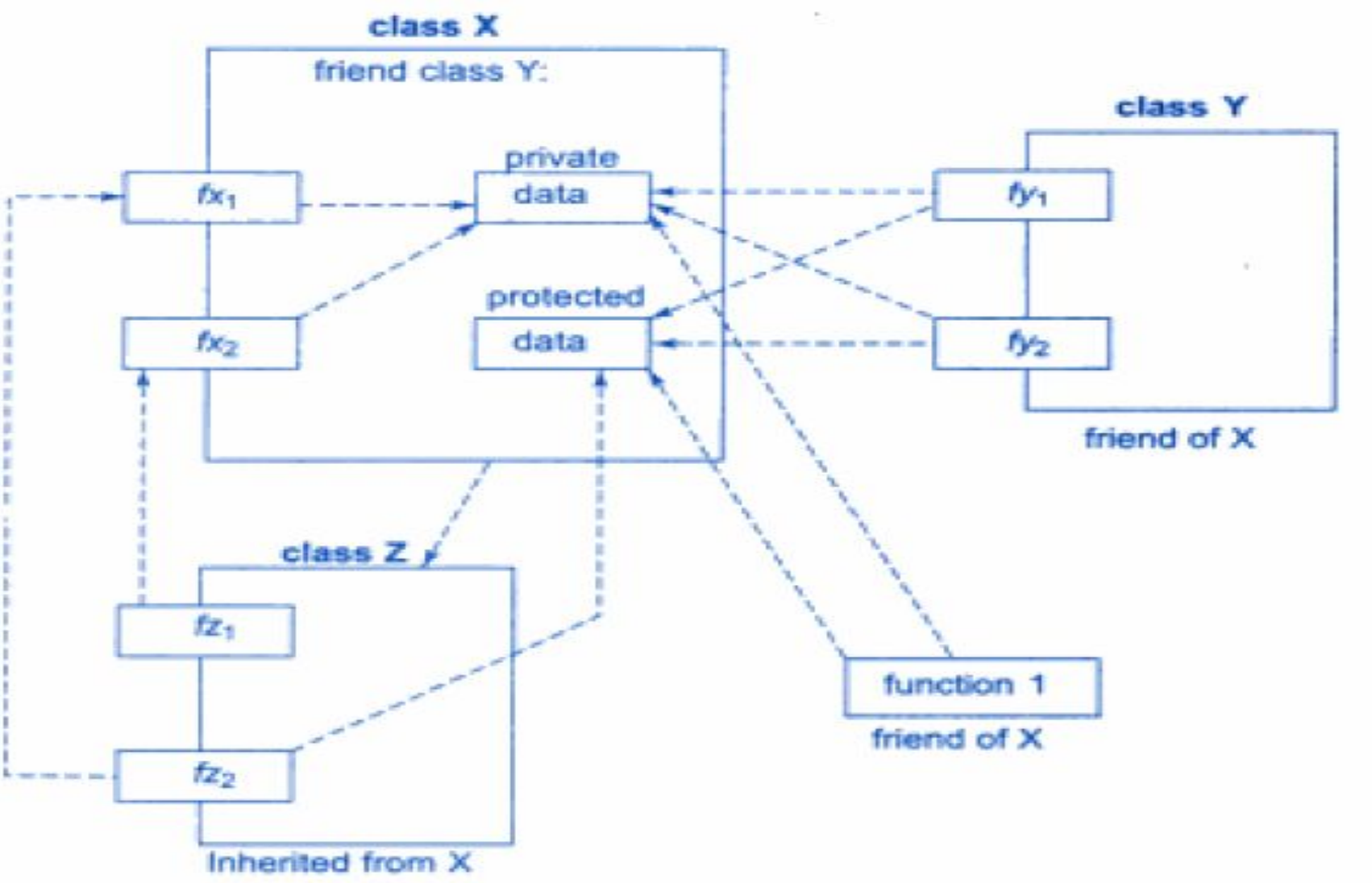
Access Control to the private and protected members of a class

- Various function that can access these members:

1. A function that is a friend of the class.
2. A member function of a friend class.
3. A member function of derived class
4. Own member functions.

*

Access Control to the private and protected members of a class



Should I use Private or Protected ?

- **Answer is "How much trust are you willing to put into the programmer of the derived class?"**
- **By default**, assume the derived class is not to be trusted, and **make your members private**. If you have a very good reason to give free access of the base class's internals to its derived classes, then you can make them protected.

*

Multilevel Inheritance

A derived class with multilevel inheritance is declared as follows:

```
class A { ……..};    // Base class
class B : public A{ ……..};// B derived from A
class C : public B{ ……..}; //C derived from B
```

This process can be extended to any number of levels.

Example

- Let us assume test result of CSE students are stored in three different classes.
- Class **student** stores roll-number, class **test** stores marks of two subjects, and class **result** contains the total marks obtained in test.

```
#include<iostream>
using namespace std;

class student
{
   protected:

      int roll_number;
   public:
      void get_number(int);
      void put_number(void);
};
```

```
void student :: get_number(int a)
{

      roll_number=a;
}

void student :: put_number()
{
      cout<<" Roll
number:"<<roll_number<<"\n";
}
```

Example

```
class test : public student
{
protected:
        float sub1;
        float sub2;
public:
        void get_marks(float, float);
        void put_marks(void);
};
```

```
void test:: get_marks(float x, float y)
{
        sub1=x;
        sub2=y;
}
void test:: put_marks()
{
        cout<<"marks in sub1 = "<<sub1<<"\n";
        cout<<"marks in sub2 = "<<sub2<<"\n";
 }
```

Example

```
class result : public test
{
    float total;
  public:
        void display(void);
};
```

```
void result::display(void)
{
    total=sub1+sub2;
    put_number();
    put_marks();
    cout<<"total = "<<total <<"\n";
}
```

Q:How many members are in class result after inheritance?

```
private:
    float total;                    // own member
protected:
    int roll_number;                // inherited from  student via test
    float sub1;                     // inherited from test
    float sub2;                     // inherited from test
public:
   void get_number(int);                // from student via test
   void put_number(void);               // from student via test
   void get_marks(float, float);        // from test
   void put_marks(void);                // from test
   void display(void);                  // own member
```

Example

```
int main()

{
      result student1;
      student1.get_number(111);
      student1.get_marks(75.0,59.5);
      student1.display();
       return 0;
}
```

Output:

Roll number:111
marks in sub1 = 75
marks in sub2 = 59.5
total = 134.5

Q:When base class is derived in public mode, then_____ .

1. public members of base class become private members of derived class.
2. public members of base class become protected members of derived class.
3. public members of base class become public members of derived class.
4. protected members of base class become protected members of derived class.
5. protected members of base class become private members of derived class.
6. protected members of base class become public members of derived class.

B. Only 3, 4

A. Only 1, 5

B. Only 3, 4

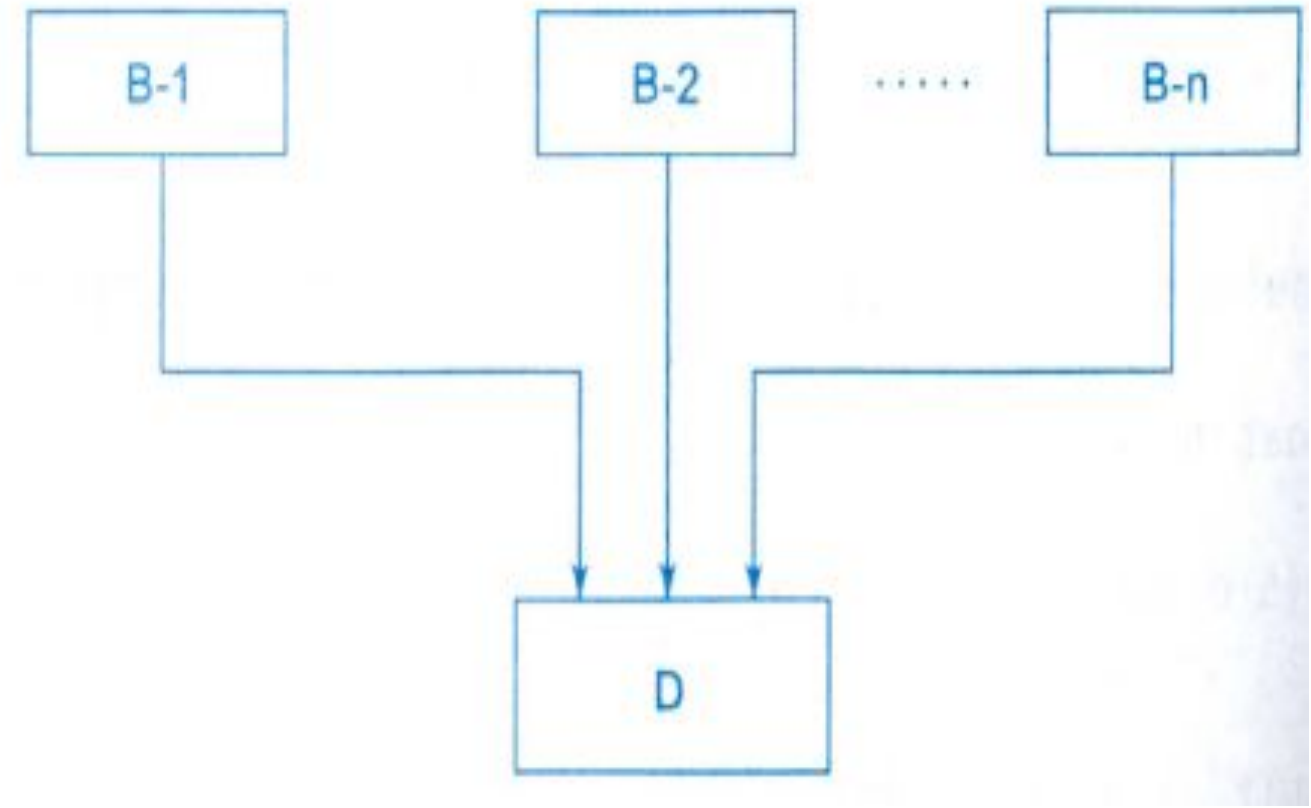
*

Multiple Inheritance

- Definition:
  A class can inherit the attributes of two or more classes. This is known as multiple inheritance.
- Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.
- The syntax of derived class with multiple base classes is as follows:

```
class D : visibility B1, visibility B2
{
   ……..
   …….. (Body of D)
   ……..
};
```

Multiple Inheritance (Cont...)



Multiple Inheritance (Cont...)

```
#include<iostream>
Using namespace std;

class M
{
    protected:
       int m;
    public:
       void get_m(int);
};

class N
{
    protected:
       int n;
    public:
       void get_n(int);
};
class P: public M, public N
{
    public:
       void display(void);
};
```

Q:How many members are in class P after inheritance?

```
class P
{
  protected:
       int m;                    // from M
       int n;                    // from N
  public:
       void get_m(int);          // from M
       void get_n(int);          // from N
       void display(void);       // own member
};
```

Multiple Inheritance (Cont...)

```
void M:: get_m(int x)
{
    m=x;
}
void N::get_n(int y)
{
     n=y;
}
void P::display(void)
{
    cout<<"m="<<m<<"\n";
    cout<<"n="<<n<<"\n";
    cout<<"m*n="<<m*n<<"\n";
}
```

```
int main()
{
    P p;
    p.get_m(10);
    p.get_n(20);
    p.display();
    return 0;
}
```

****OUTPUT****

m=10

n=20

m*n=200

Function Overriding

- Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class.
- In other words, it is the redefinition of base class function in its derived class with same signature and return type.
- In function overriding, if we call overridden function using the object of the derived class, the function of the derived class is executed.

*

Function Overriding

consider the following situation:

```
class base
{
    public :
      void display( )
      {
       cout<< base class\n";
      }
};
class derived: public base
{
    public :
      void display( )
      {
       cout<< "derived class\n";
      }
};
```

```
void main()
{
derived b;   //derived
   class object
b.display();
}
******OUTPUT******
derived class
```

Ambiguity Resolution in Inheritance

- Consider following two classes:

```
class M
{   public:
        void display()
        {
            cout<<“class M\n”;
        }

};
```

```
class N
{
public:
        void display()
        {
            cout<<“class N\n”;
        }
};
```

```
class P: public M,public N
{   public:
        void display()
        {
            display();//ambiguity
        }
};
```

```
class P: public M,public N
{   public:
        void display()
        {
            M::display();
        }
};
```

*

How to access base class overridden function?

Using scope resolution operator;

```
class base
{
    public :
      void display( )
      {
       cout<< base class\n";
      }
};
class derived: public base
{
    public :
      void display( )
      {
       cout<< "derived class\n";
      }
};
```

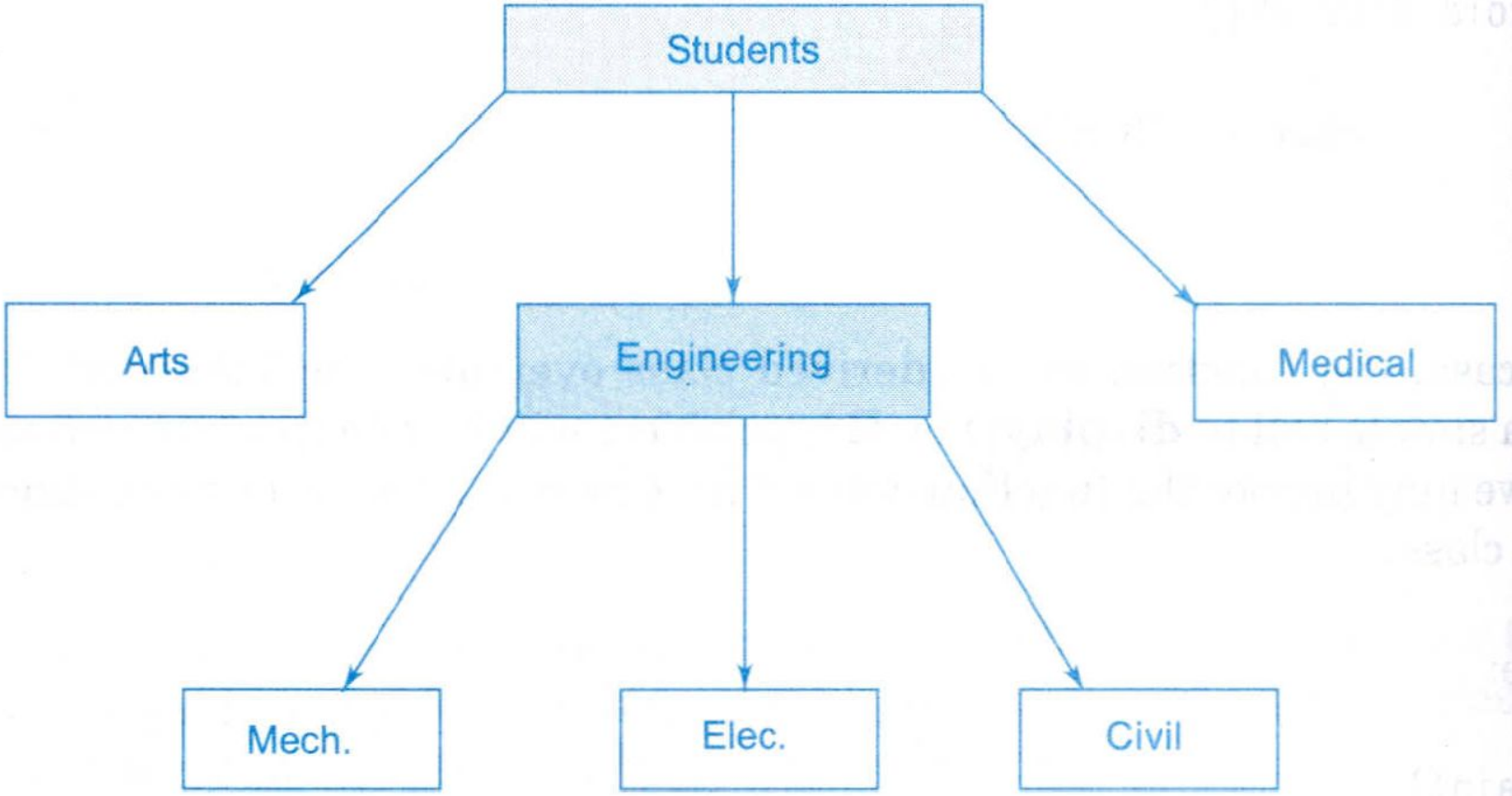
```
void main()
{
derived b;   //derived class
   object
b.display(); //calls display of
   derived
b.base::display(); //calls display of
   base
b.derived::display(); //calls
   display of derived
}
******OUTPUT******
derived class
Base class
derived class
```

Overloading vs. overriding

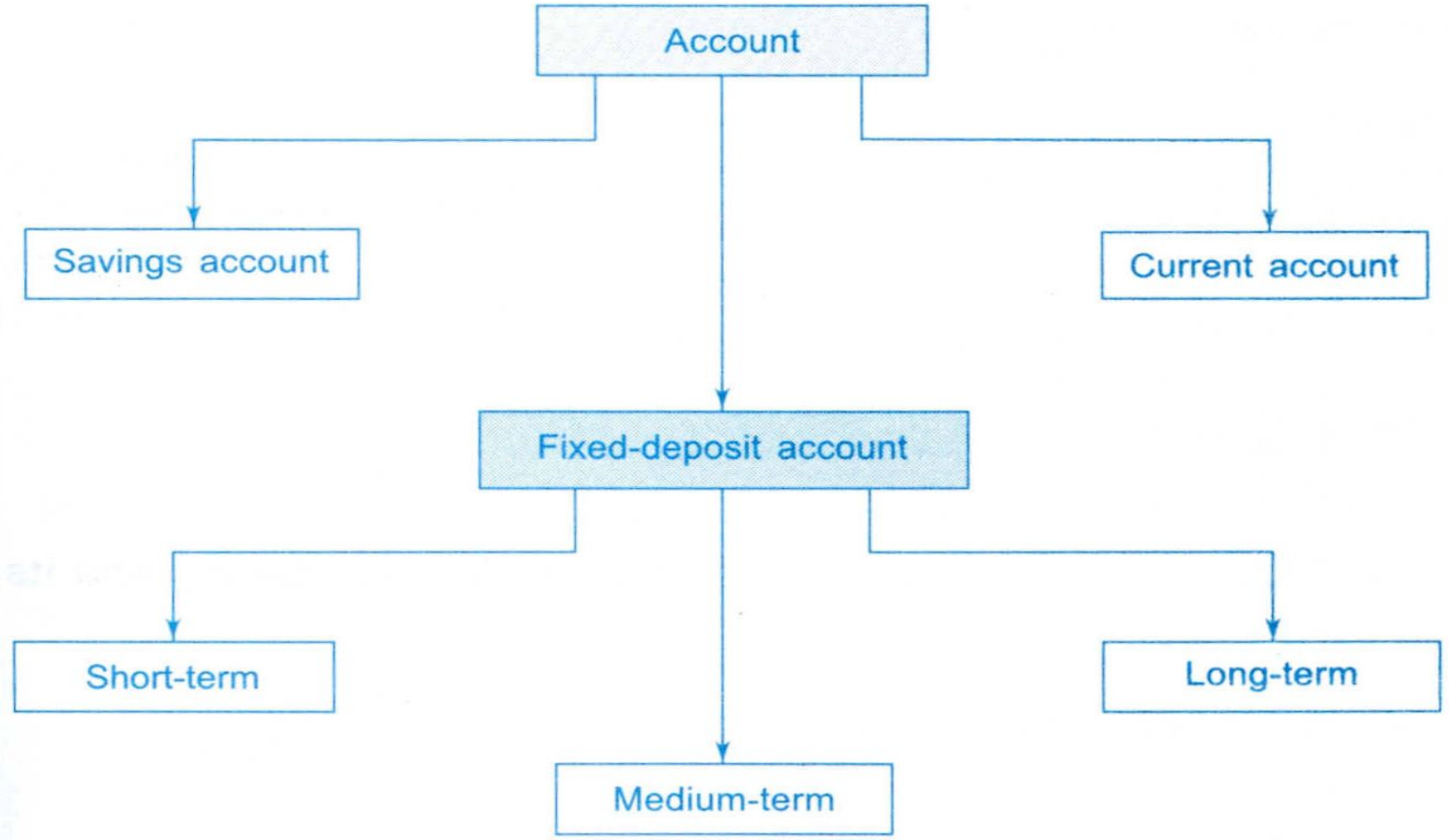
1. Overriding of functions occurs when one class is inherited from another class. Overloading occurs without inheritance.
2. Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
3. Overridden functions are in different scopes; whereas overloaded functions are in same scope.
4. Overriding is needed when derived class function has to do some added or different job than the base class function

Hierarchical Inheritance

For hierarchical design of program we can use hierarchical inheritance.



Another Example



Question

Write a class number that stores a private data member ***num***(int type) and two member functions namely **getNumber()** and **returnNumber()**. From this, derive the classes square and cube.

Sample output:

Enter an integer number: 10

Square of 10 is: 100

Enter an integer number: 20

Cube of 20 is: 8000

*

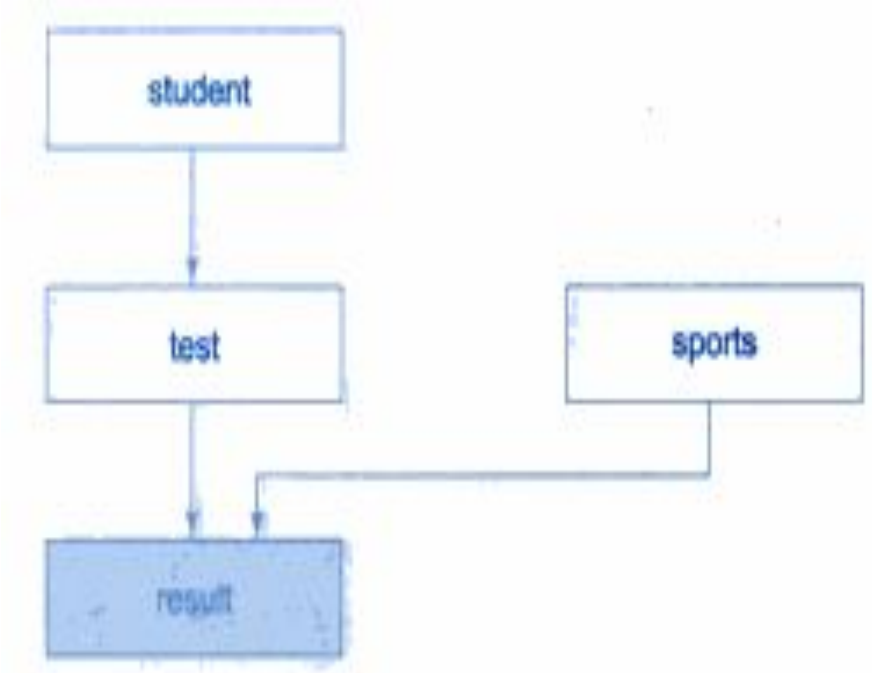
```
#include <iostream>
using namespace std;
 class Number
{
    private:
        int num;
    public:
        void getNumber(void)
        {
            cout << "Enter an integer
    number: ";
            cin  >> num;
        }
        //to return num
        int returnNumber(void)
        { return num; }
};
```

```
class Square:public Number
{  public:
    int getSquare(void)
    {
        int num,sqr;
        num=returnNumber();
    //get number
          from class Number
        sqr=num*num;
        return sqr;
    }
};
```

```
//Class Cube, to calculate cube of a
    number
class Cube:public Number
{
    private:

    public:
    int getCube(void)
    {
        int num,cube;
        num=returnNumber(); //get
    number from              class
    Number
        cube=num*num*num;
        return cube;
    }
};
```

```
int main()
{
        Square objS;
        Cube objC;
        int sqr,cube;

        objS.getNumber();
        sqr =objS.getSquare();
        cout << "Square of "<<
    objS.returnNumber()       << " is: " <<
    sqr  << endl;

        objC.getNumber();
        cube=objC.getCube();
        cout << "Cube   of "<<
    objS.returnNumber()       << " is: " <<
    cube << endl;
         return 0;
}
```

HYBRID Inheritance




```
#include<iostream>
using namespace std;
class student
{
      protected:
     int roll_number;
   public:
     void get_number(int a)
     {
          roll_number=a;
     }
     void put_number(void)
     {
          cout<<" Roll number:"<<roll_number<<"\n";
     }

};
```

HYBRID Inheritance (cont…)

```
class test : public student
{
    protected:
      float part1,part2;
    public:
    void get_marks(float x, float y)
    {  part1=x;
      part2=y;
    }
    void put_marks(void)
    {    cout<<"marks obtained =
    "<<"\n";
          cout<<"part1 =
    "<<part1<<"\n";
          cout<<"part2 =
    "<<part2<<"\n";
    }
};
```

```
class sports
{
        protected:
                float score;
         public:
        void get_score(float s)
        {      score= s;
        }
        void put_score(void)
        {
        cout<<"Sports wt:"<<score<<"\n\n";
        }
};
```

HYBRID Inheritance (cont…)

```
class result : public test,public sports
{
 float total;
 public:
 void display(void);
 };
 void result::display(void)
 {
    total=part1+part2+score;
    put_number();
    put_marks();
    put_score();
    cout<<"total score: =  "<<total <<"\n";
 }
```

```
int main()
{
    result student1;
    student1.get_number(1234
    );
    student1.get_marks(27.5,3
    3.0);
    student1.get_score(6.0);
    student1.display();
    return 0;
}
```

Output:
Roll number:123
marks obtained :
part1 = 27.5
part2 = 33
Sports wt:6

total score: = 66.5