

## Unit-III

### 7. Inheritance

Extending Classes, Concept of inheritance, Base class, Derived class, defining derived classes, Visibility modes: Private, public, protected; Single inheritance: Privately derived, publicly derived; Making a protected member inheritable, Access Control to private and protected members by member functions of a derived class, Multilevel inheritance, Nesting of classes.

---

#### ➤ Concept of inheritance

C++ supports reusability through inheritance. “The mechanism of deriving a new class from old one is called inheritance or derivation”.

The old class is referred as **base class** and the new class is called as **derived class or subclass**.

#### ➤ Defining derived class

A derived class can be defined by specifying its relationship with the base class.

The syntax of defining a derived class is: --

```
class derived_class_name: visibility_mode base_class_name
{
    //members of derived class
};
```

Here,

- The colon indicates that the derived\_class\_name is derived from the base\_class\_name.
- The visibility\_mode is optional and if present, may be either private or public.
- The default visibility mode is private. The visibility mode specifies whether the features of the base class are privately derived or publicly derived.

#### Example:

```
class D: private B                //private derivation
{
    //members of D
};
```

```
class D: public B                 //public derivation
{
    //members of D
};
```

```
3. class D: protected B          //protected derivation
{
    //members of D
};
```

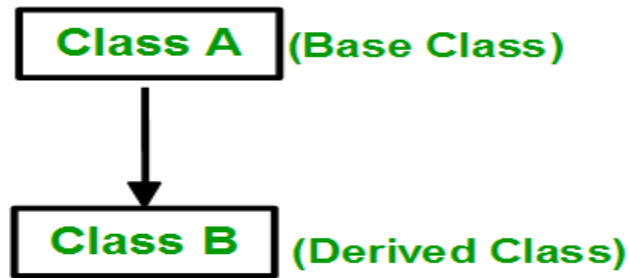
```
4. class D: B                     //private derivation by default
{
    //members of D
};
```

➤ **Types of inheritance**

1. Single or single level inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

**1. Single Inheritance**

In single inheritance, one subclass is derived from one base class only.



**Syntax:**

```
class derived_class_name:visibility_mode base_class
{
    Members of derived class
};
```

**Example:**

```
class B
{
    ... ..
};
class D: public B
{
    ... ..
};
```

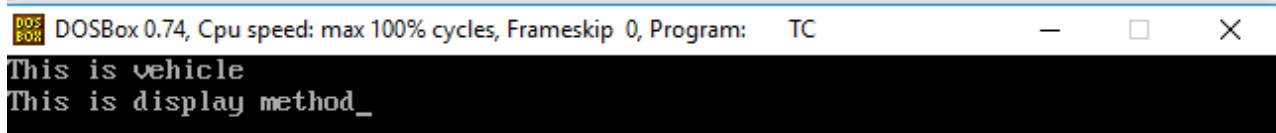
**//Program for single inheritance**

```
#include<iostream.h>
#include<conio.h>
class vehicle
{
public:
    vehicle()
    {
        cout<<"This is vehicle"<<endl;
    }
};
class car:public vehicle
{
public:
    void display()
    {
        cout<<"This is display method";
    }
};
```

```

};
void main()
{
clrscr();
car c; //calls and executes constructor of vehicle class
c.display();
getch();
}

```



- **Single inheritance: Privately derived, publicly derived; Making a protected member inheritable**  
**OR**  
**Visibility Mode in Single Inheritance**

The visibility mode specifies the control over the inherited members within the derived classes. A class can inherit a base class in three visibility modes in Single Inheritance:

### 1. Public derivation

If a derived class is inherited from a base class publicly, then the public members of the base class will become public in the derived class and protected members of the base class will become protected in the derived class. While the private members will remain inaccessible by the derived class.

**The following program code shows how to apply the public visibility mode in a derived class in Single Inheritance:**

```

class base_class
{
    // data members
    // member functions
};
class derived_class: public base_class
{
    // data members
    // member functions
};

```

### Example:

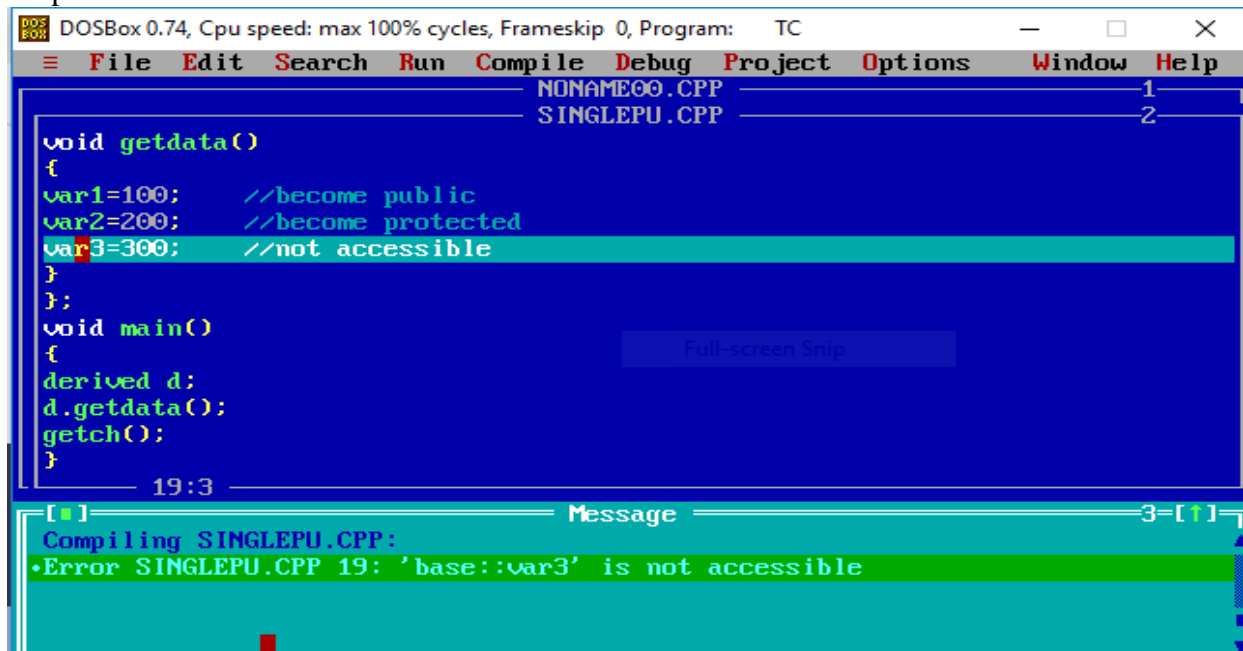
The following program code displays the working of public visibility mode with all three access specifiers of the base class:

### //Single level inheritance: public derivation

```
#include<iostream.h>
#include<conio.h>
class base
{
public:
    int var1;
protected:
    int var2;
private:
    int var3;
};

class derived:public base
{
public:
void getdata()
{
var1=100; //accessible and become public
var2=200; //accessible and become protected
var3=300; //not accessible
}
};
void main()
{
derived d;
d.getdata();
getch();
}
```

Output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP 1
SINGLEPU.CPP 2

void getdata()
{
var1=100: //become public
var2=200: //become protected
var3=300: //not accessible
}
};
void main()
{
derived d;
d.getdata();
getch();
}
19:3

Message 3
Compiling SINGLEPU.CPP:
Error SINGLEPU.CPP 19: 'base::var3' is not accessible
```

In the above example, the visibility of the derived class `derived_class` is set as `public` so all the access specifiers will remain the same. `var1` is a public member, so it is accessible outside the derived class. `var2` is a protected member, so it is accessible within the derived class but not outside that. Whereas, `var3` is a private member so it is not accessible outside the base class `base_class`.

## 2. Private derivation

If a derived class is inherited from a base class privately, then both the public and protected members of the base class will become private in the derived class whereas, the private members will remain inaccessible by the derived class.

### Syntax:

```
class base_class
{
    // data members
    // member functions
};
class derived_class: private base_class
{
    // data members
    // member functions
};
```

### Example:

The following code displays the working of private visibility mode with all three access specifiers of the base class:

#### //Single inheritance: private derivation

```
#include<iostream.h>
#include<conio.h>
class base
{
public:
    int var1;
protected:
    int var2;
private:
    int var3;
};
class derived:private base
{
public:
    void getdata()
    {
        var1=105; //accessible and become private
        var2=205; //accessible and become private
        var3=305; //not accessible
    }
};
```

```

void main()
{
derived d;
d.getdata();
getch();
}

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

File Edit Search Run Compile Debug Project Options Window Help

NONAME00.CPP 1  
SINGLEPU.CPP 2  
SINGLEPR.CPP 5

```

void getdata()
{
var1=105; //accessible and become private
var2=205; //accessible and become private
var3=305; //not accessible
};
void main()
{
derived d;
d.getdata();
}

```

23:3

Message 3=11

Compiling SINGLEPR.CPP:  
•Error SINGLEPR.CPP 23: 'base::var3' is not accessible

F1 Help Space View source Edit source F10 Menu

### 3. Protected derivation

If a derived class is inherited from a base class protectedly, then both public and protected members of the base class will become protected in the derived class whereas the private members will remain inaccessible by the derived class.

#### Syntax:

```

class base_class
{
    // data members
    // member functions
};
class derived_class: protected base_class
{
    // data members
    // member functions
};

```

#### Example:

The following program code shows how to apply the protected visibility mode in a derived class in Single Inheritance:

# //Single inheritance:protected derivation

```
#include<iostream.h>
#include<conio.h>
class base
{
public:
    int var1;
protected:
    int var2;
private:
    int var3;
};

class derived:protected base
{
public:
void getdata()
{
var1=101; //accessible and become protected
var2=201; //accessible and become protected
var3=301; //not accessible
}
};

void main()
{
derived d;
d.getdata();
getch();
}
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP 1
SINGLEPU.CPP 2
SINGLEPR.CPP 5

void getdata()
{
var1=101; //accessible and become protected
var2=201; //accessible and become protected
var3=301; //not accessible
};
void main()
{
derived d;
d.getdata();
getch();
}

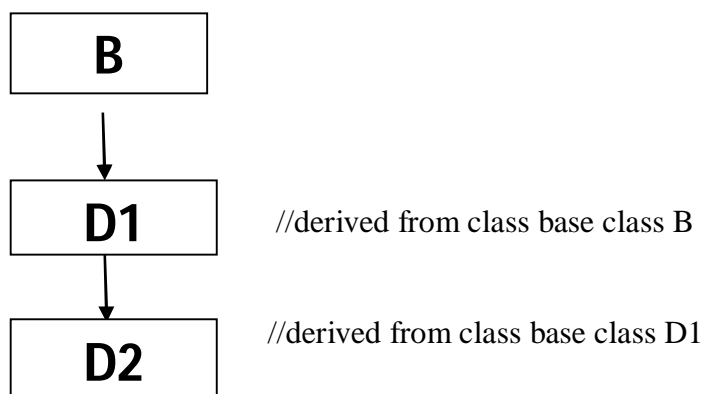
Message 3-[F1]
Compiling SINGLEPR.CPP:
Error SINGLEPR.CPP 21: 'base::var3' is not accessible
F1 Help Space View source F10 Menu
```

The following table illustrates the control of the derived classes over the members of the base class in different visibility modes in single inheritance:

Base Class	Derived Class	Derived Class	Derived Class
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Inherited / Remains Private	Not Inherited / Remains Private	Not Inherited / Remains Private

## 2. Multilevel Inheritance

In this type of inheritance, a derived class is created from another derived class.



### Syntax:

```

class B
{
... ..
};
class D1: public B
{
... ..
};
class D2: public D1
{
... ..
};
  
```



### //Program for multilevel inheritance

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;
char name[20];
public:
void get_rn()
{
cout<<"Enter roll no. and name="; cin>>rno>>name;
}
void put_rn()
{
cout<<"\n Roll No.="<<rno<<endl; cout<<"Name="<<name<<endl;
}
};
class marks: public student
{
protected:
int m1,m2,m3;
public:
void get_marks()
{
cout<<"Enter marks of m1,m2 and m3="; cin>>m1>>m2>>m3;
}
};
class game: public marks
{
protected:
int gm,total;
public:
void calculate()
{
cout<<"Enter the marks of game="; cin>>gm;
total=m1+m2+m3+gm;
}
void display()
{
put_rn();
cout<<"Total="<<total;
}
};
void main()
{
clrscr();
game g;
g.get_rn();
g.get_marks();
g.calculate();
g.display();
getch();
}
```

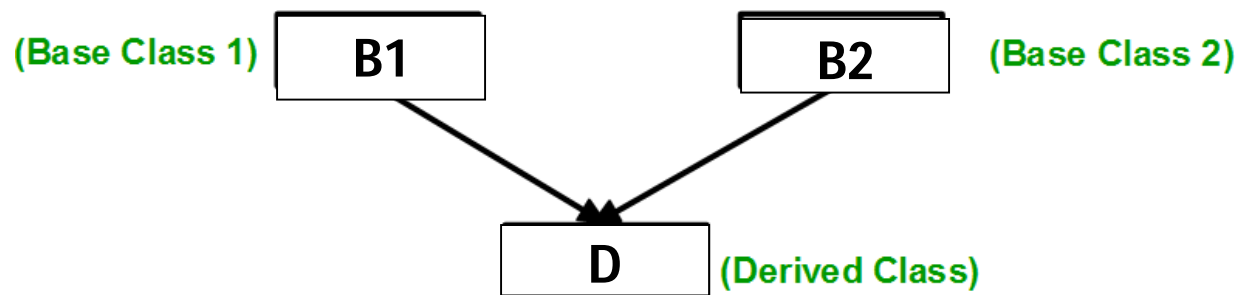
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter roll no. and name=101 Bjarne
Enter marks of m1,m2 and m3=75 83 90
Enter the marks of game=91

Roll No.=101
Name=Bjarne
Total=339_
```

### 3. Multiple Inheritance

In multiple Inheritance, a derived class have more than one base class. It means that a derived class can get the members from different base classes.

It has the following structure:



**Syntax of a derived class with multiple base classes are as follows:**

```
class D: visibility_mode B1, visibility_mode B2,...
{
//Members of D
};
```

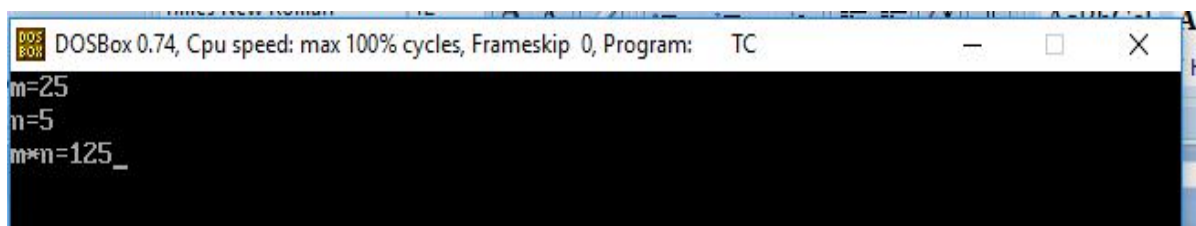
**Example:**

```
class B1
{
... ..
};
class B2
{
... ..
};
class D: public B1, public B2
{
//Members of D
};
```

Here, the number of base classes will be separated by a comma (',') and the access mode for every base class must be specified.

**Example:****// Program for multiple inheritance**

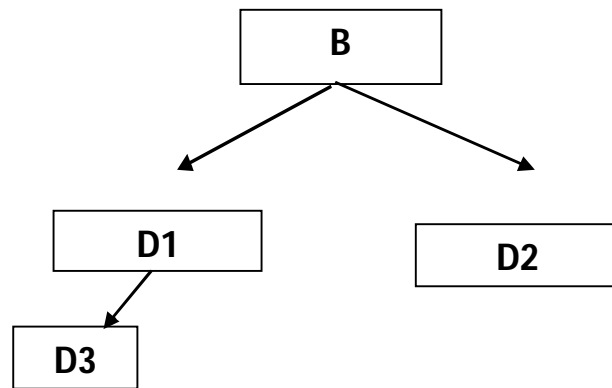
```
#include<iostream.h>
#include<conio.h>
class B1
{
protected:
int m;
public:
void get_m(int x)
{
m=x;
}
};
class B2
{
protected:
int n;
public:
void get_n(int y)
{
n=y;
}
};
class D: public B1, public B2
{
public:
void display()
{
cout<<"m="<<m<<endl;
cout<<"n="<<n<<endl;
cout<<"m*n="<<m*n;
}
};
void main()
{
clrscr();
D d;
d.get_m(25);
d.get_n(5);
d.display();
getch();
}
```

A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window contains a black terminal area with white text showing the output of the program: "m=25", "n=5", and "m\*n=125\_". The cursor is positioned at the end of the last line.

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
m=25
n=5
m*n=125_
```

#### 4. Hierarchical Inheritance

In this type of inheritance, more than one derived class is created/derived from a single base class.



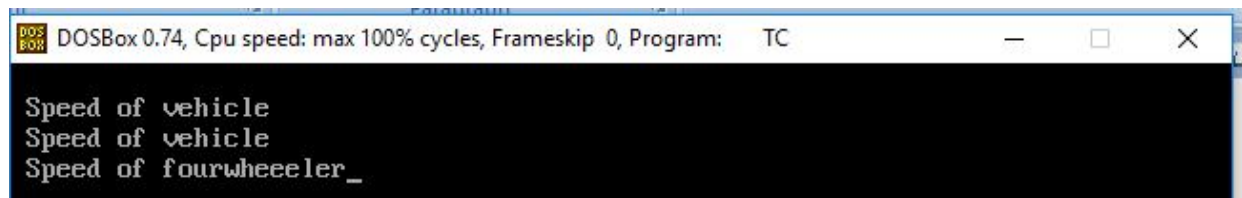
##### Syntax:-

```
class B
{
// members of B
};
class D1: public B
{
// members of D1
};
class D2: public B
{
// members of D2
}
class D3: public D1
{
// members of D3
}
```

## //Program for hierarchical inheritance

```
#include<iostream.h>
#include<conio.h>
class vehicle
{
public:
virtual void speed()
{
cout<<"\n Speed of vehicle";
}
};
class twowheeler:public vehicle
{
public:
void speed()
{
cout<<"\n Speed of vehicle";
}
};
class fourwheeler:public vehicle
{
public:
void speed()
{
cout<<"\n Speed of fourwheeler";
}
};
void main()
{
clrscr();
vehicle *p,v;
p=&v;
p->speed();
twowheeler t;
p=&t;
```

```
p->speed();  
fourwheeler f;  
p=&f;  
p->speed();  
getch();  
}
```

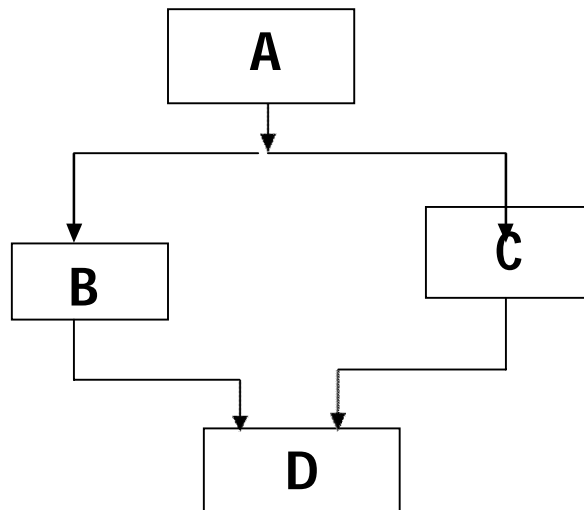


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Speed of vehicle  
Speed of vehicle  
Speed of fourwheeler_
```

## 5. Hybrid Inheritance

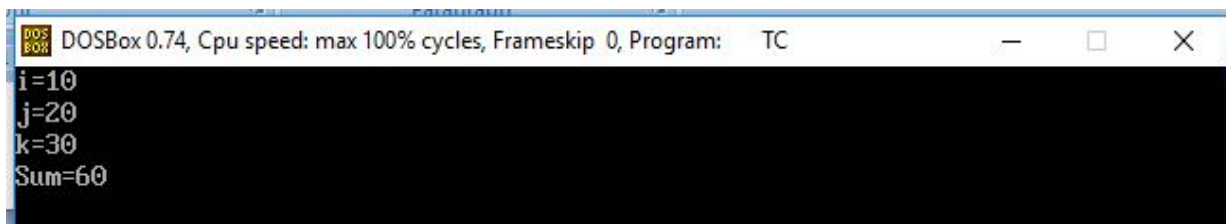
It is the combination of more than one type of inheritance to design a program.  
It has the following structure:



```

//Program for hybrid inheritance
#include<iostream.h>
#include<conio.h>
class base
{
public:
int i;
};
class derived1:virtual public base
{
public:
int j;
};
class derived2:virtual public base
{
public:
int k;
};
class derived3:public derived1,public derived2
{
public:
int sum;
};
void main()
{
clrscr();
derived3 d;
d.i=10;
d.j=20;
d.k=30;
d.sum=d.i+d.j+d.k;
cout<<"i="<<d.i<<endl;
cout<<"j="<<d.j<<endl;
cout<<"k="<<d.k<<endl;
cout<<"Sum="<<d.sum;
getch();
}

```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```

i=10
j=20
k=30
Sum=60

```

### ➤ Nesting of classes

- A class within a class is known as nested class.
- It means that a class can also contain another class definition inside itself, which is called “Inner Class” in C++.
- In this case, the containing class is referred to as the “Enclosing Class”.
- The Inner Class definition is considered to be a member of the Enclosing Class.
- The inner class has the same access rights as any other member of the class.
- The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.

#### //Program for nested class

```
#include<iostream.h>
#include<conio.h>
class outer
{
public:
    void show() //member function of outer class
    {
        i.display();
    }
    class inner //member of outer class
    {
    public:
        void display()
        {
            cout<<"Display function of inner class";
        }
    }; //object of inner class
};
void main()
{
    clrscr();
    outer o;
    o.show();
    getch();
}
```

