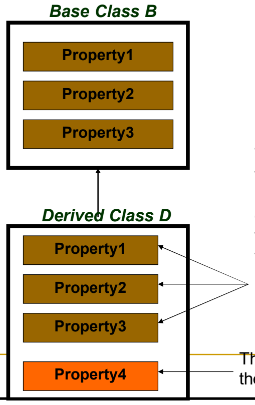
Inheritance

Dr.Pooja Jain
IIIT Nagpur

INHERITANCE

- Inheritance is the process of creating new classes from the existing class or classes.
- Using inheritance, one can create general class that defines traits common to a set of related items. This class can then be inherited (reused) by the other classes by using the properties of the existing ones with the addition of its own unique properties.
- The old class is referred to as the **base class** and the new classes, which are inherited from the base class, are called **derived classes**.

INHERITANCE



The derived class D also contains the same three properties as those in the base class B. Instead of defining these properties again, they can be reused or inherited from the base class B.

This is the unique property of the derived class D.

Forms of Inheritance

Single Inheritance	Multiple Inheritance	Multilevel Inheritance	Hierarchical Inheritance
If a class is derived from a single base class, it is called as single inheritance.	If a class is derived from more than one base class, it is known as multiple inheritance	The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.	If a number of classes are derived from a single base class, it is called as hierarchical inheritance

A derived class can be defined as follows:

```
class derived_class_name : access_specifier
   base_class_name
{
   data members of the derived class ;
   member functions of the derived class ;
   }
```

- The colon (**:**), indicates that the class *derived_class_name* is derived from the class *base_class_name.*
- The *access_specifier* may be **public, private** or **protected** (will be discussed further).
- If no *access_specifier* is specified, it is **private** by default.
- The access_specifier indicates whether the members of the base class are **privately derived** or **publicly derived.**

Public inheritance

- When a derived class **publicly** inherits the base class, all the **public members** of the base class also become **public** to the derived class and the **objects** of the derived class can access the **public members** of the base class.
- The following **program** will illustrate the use of the **single inheritance.** This program has a base class **B**, from which class **D** is inherited **publicly**.

Example:

```
#include<iostream.h>
class Rectangle
{
   private:
float length ; // This can't be inherited
   public:
float breadth ; // The data and member functions are inheritable
void Enter_lb(void)
{
     cout << "\n Enter the length of the rectangle : ";
            cin >> length ;
     cout << "\n Enter the breadth of the rectangle : ";
     cin >> breadth ;
}
float get_l(void)
{    return length ; }
}; // End of the class definition
```

This member function is used to get the value of data member ‘length’ in the derived class

Cont.

```
class Rectangle1 : public Rectangle
{
   private:
float area ;
     public:
void Rec_area(void)
{  area = get_l( ) * breadth ;  }
// area = length * breadth ;  can't be used here
```

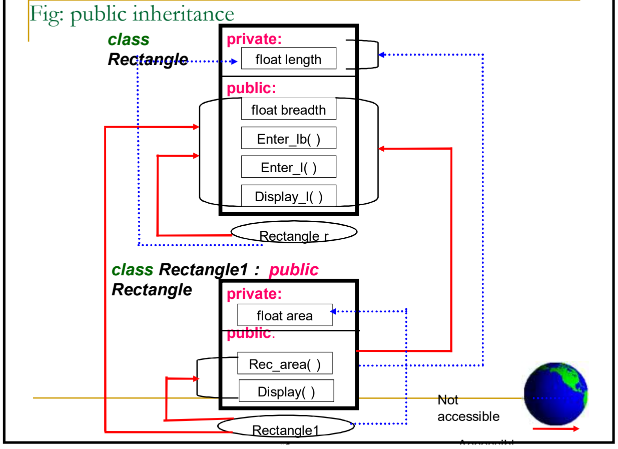
The base class is publicly inherited by the derived class. Thus all the public members of the base class can be inherited by the derived class.

*The data member “**length”** of the base class can’t be inherited as it is defined in **private** mode. Thus the member function **get_l()** is used here which is declared in the **public** mode in the base class.*

Cont.

```
  void Display(void)
  {
   cout << "\n Length = " << get_l( ) ; // Object of the derived class can't
  //    inherit the private member of the base class. Thus the member
  //   function is used here to get the value of data member 'length'.
   cout << "\n Breadth = " << breadth ;
   cout << "\n Area = " << area  ;
  }
};  // End of the derived class definition D
void main(void)
{
Rectangle1 r1 ;
r1.Enter_lb( );
r1.Rec_area( );
r1.Display( );
}
```

Fig: public inheritance



class Rectangle

private:

float length

public:

float breadth

Enter_lb( )

Enter_l( )

Display_l( )

Rectangle r

class Rectangle1 : public Rectangle

private:

float area

public:

Rec_area( )

Display( )

Rectangle1

Not accessible

Private inheritance

- When a derived class **privately** inherits a base class, all the **public members** of the base class become **private** for the derived class.
- In this case, the **public members** of the base class can only be accessed by the **member functions** of the derived class.
- The **objects** of the derived class cannot access the **public members** of the base class.
  - ***Note that whether the derived class is inherited publicly or privately from the base class, the private members of the base class cannot be inherited.***

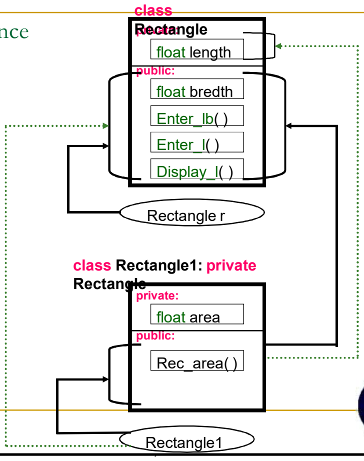
Example:

```
  #include<iostream.h>
  #include<conio.h>
  class Rectangle
  {
        int length, breadth;
        public:
                void enter()
                {
                        cout << "\n Enter length: "; cin >> length;
                        cout << "\n Enter breadth: "; cin >> breadth;
                }
                int getLength()
                {
                        return length;
                }
                int getBreadth()
                {
                        return breadth;
                }
                void display()
                {
                        cout << "\n Length= " << length;
                        cout << "\n Breadth= " << breadth;
                }
  };
```

Cont.

```
class RecArea : private Rectangle
{

    public:
          void area_rec()
          {
                    enter();
                    cout << "\n Area = " << (getLength() * getBreadth());
          }
};
void main()
{
    clrscr();
    RecArea r ;
    r.area_rec();
    getch();
}
```

private inheritance



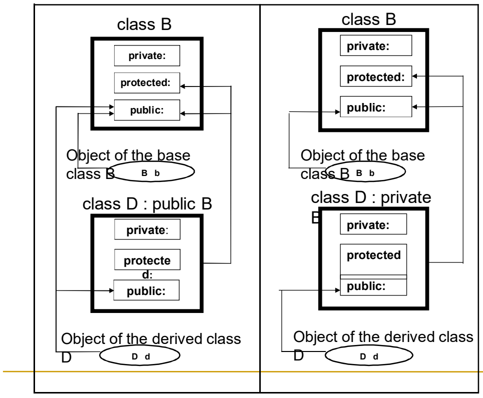
The protected access specifier

- The third access specifier provided by **C++** is **protected**.
- The members declared as **protected** can be accessed by the **member functions** within their own class and any other class immediately derived from it.
- These members cannot be accessed by the functions outside these two classes.
- Therefore, the **objects** of the derived class cannot access **protected members** of the base class.
- When the **protected members** (data, functions) are inherited in **public mode**, they become **protected** in the derived class. Thus, they can be accessed by the **member functions** of the derived class.
- On other hand, if the **protected members** are inherited in the **private mode,** the members also become **private** in the derived class.
- They can also be accessed by the member functions of the derived class, but cannot be inherited further.

Table: Accessibility

Access Specifier	Accessibility from own class	Accessibility from derived class	Accessibility from objects outside the base class
public	Valid	Valid	Valid
protected	Valid	Valid	Not valid
private	Valid	Not valid	Not valid

Fig: Access specifier with inheritance



Overriding the member functions

- The member functions can also be used in a derived class, with the same name as those in the base class.
- One might want to do this so that calls in the program work the same way for objects of both base and derived classes.
- The following **program** will illustrate this concept.

Example: Overriding of member function

```
#include<iostream.h>
const int len = 20 ;
class Employee
{
    private:
            char F_name[len];
            int I_number ;
            int age ;
            float salary ;
    public:
            void Enter_data(void)
            {
                        cout << "\n Enter the first name = " ; cin >> F_name ;
                        cout << "\n Enter the identity number = " ; cin >> I_number ;
                        cout << "\n Enter the age = " ; cin >> age ;
                        cout << "\n Enter the salary = " ; cin >> salary ;
            }
            void Display_data(void)
            {
                        cout << "\n Name = " << F_name ;
                        cout << "\n Identity Number = " << I_number ;
                        cout << "\n Age = " << age ;
                        cout << "\n Salary = " << salary ;
            }
};  // End of the base class
```

Cont.

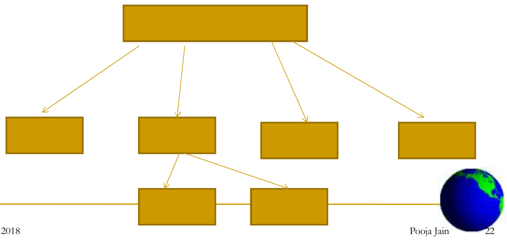
```
class Engineer : public Employee
{
    private:
            char design[len] ; // S_Engineer, J_Engineer, Ex_Engineer etc

    public:
            void Enter_data( )
            {
                        Employee :: Enter_data( ) ;  // Overriding of the member function
                        cout << "\n Enter the designation of the Engineer: " ; cin >>
    design ;
            }


            void Display_data(void)
            {
                        cout << "\n *******Displaying the particulars of the Engineer**** \n"
    ;
                        Employee :: Display_data( ) ; // Overriding of the member function
                        cout << "\n Designition = " << design ;
            }
}; // End of the derived class
```

Cont.

```
void main(void)
{
        Engineer er ;
        er.Enter_data( ) ;
        er.Display_data( );
}
```

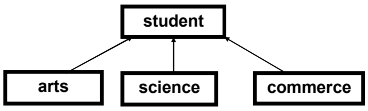
Hierarchical

- Refers to systems that are organized in the shape of a pyramid, with each row of objects linked to objects directly beneath it. Hierarchical systems pervade everyday life.



Hierarchical Inheritance:

When two or more classes are derived from a single base class, then Inheritance is called the hierarchical inheritance.



Student is a base class, from which the three classes viz. **arts, science** and **commerce** have been derived.

Example:

```
include<iostream.h>
const int len = 20 ;
class student  //  Base class
{
    private: char F_name[len] ,  L_name[len] ;
             int age,  int roll_no ;
    public:
          void Enter_data(void)
          {
                cout << "\n\t Enter the first name: " ; cin >> F_name ;
                cout << "\t Enter the second name: "; cin >> L_name ;
                cout << "\t Enter the age: " ; cin >> age ;
                cout << "\t Enter the roll_no: " ; cin >> roll_no ;
          }
          void Display_data(void)
          {
                cout << "\n\t First Name = " << F_name ;
                cout << "\n\t Last Name = " << L_name ;
                cout << "\n\t Age = " << age ;
                cout << "\n\t Roll Number = " << roll_no ;
          }
};
```

Cont.

```
class arts : public student
{
    private:
          char asub1[len] ;
          char asub2[len] ;
          char asub3[len] ;
    public:
          void Enter_data(void)
          {
                    student :: Enter_data( );
                    cout << "\t  Enter the subject1 of the arts student: "; cin >> asub1 ;
                    cout << "\t  Enter the subject2 of the arts student: "; cin >> asub2 ;
                    cout << "\t  Enter the subject3 of the arts student: "; cin >> asub3 ;
          }
          void Display_data(void)
          {
                    student :: Display_data( );
                    cout << "\n\t Subject1 of the arts student = " << asub1 ;
                    cout << "\n\t Subject2 of the arts student = " << asub2 ;
                    cout << "\n\t Subject3 of the arts student = " << asub3 ;
          }
};
```

Cont.

```
class science : public student
{
    private:
          char ssub1[len] ;
          char ssub2[len] ;
          char ssub3[len] ;
    public:
          void Enter_data(void)
          {
                    student :: Enter_data( );
                    cout << "\t Enter the subject1 of the science student: "; cin >>
    ssub1;
                    cout << "\t Enter the subject2 of the science student: "; cin >>
    ssub2;
                    cout << "\t Enter the subject3 of the science student: "; cin >>
    ssub3;
          }
          void Display_data(void)
          {
                    student :: Display_data( );
                    cout << "\n\t Subject1 of the science student = " << ssub1 ;
                    cout << "\n\t Subject2 of the science student = " << ssub2 ;
                    cout << "\n\t Subject3 of the science student = " << ssub3 ;
          }
```

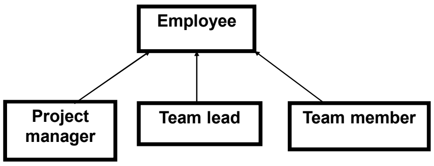
Cont.

```
class commerce : public student
{
    private:   char csub1[len], csub2[len], csub3[len] ;
    public:
          void Enter_data(void)
          {
                    student :: Enter_data( );
                    cout << "\t Enter the subject1 of the commerce student: ";
                    cin >> csub1;
                    cout << "\t Enter the subject2 of the commerce student: ";
                    cin >> csub2 ;
                    cout << "\t Enter the subject3 of the commerce student: ";
                    cin >> csub3 ;
          }
          void Display_data(void)
          {
                    student :: Display_data( );
                    cout << "\n\t Subject1 of the commerce student = " <<
    csub1 ;
                    cout << "\n\t Subject2 of the commerce student = " <<
    csub2 ;
                    cout << "\n\t Subject3 of the commerce student = " <<
    csub3 ;
```

Cont.

```
void main(void)
{
          arts a ;
          cout << "\n Entering details of the arts student\n" ;
          a.Enter_data( );
          cout << "\n Displaying the details of the arts student\n" ;
          a.Display_data( );
          science s ;
          cout << "\n\n Entering details of the science student\n" ;
          s.Enter_data( );
          cout << "\n Displaying the details of the science student\n" ;
          s.Display_data( );
          commerce c ;
          cout << "\n\n Entering details of the commerce student\n" ;
          c.Enter_data( );
          cout << "\n Displaying the details of the commerce
    student\n";
          c.Display_data( );
}
```

Hierarchical Inheritance:


Employee

Project manager

Team lead

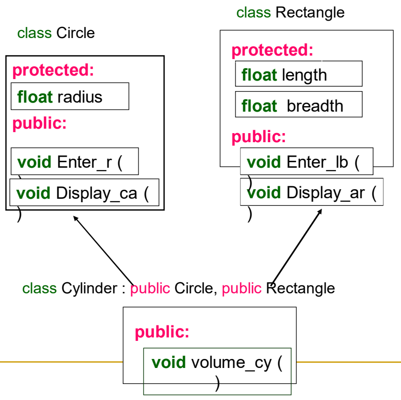
Team member

Multiple Inheritance

- When a class is inherited from more than one base class, it is known as **multiple inheritance**.
- The syntax for defining a subclass, which is inheriting more than one classes is:

```
class Subclass : access_specifier   Baseclass1,
                 access_specifier   Baseclass2,  ………
                   ……..   access_specifier   Baseclass_n
{
        members of the derived class ;
};
```

Multiple Inheritance


class Circle

protected:

float radius

public:

void Enter_r ( )

void Display_ca ( )

class Rectangle

protected:

float length

float breadth

public:

void Enter_lb ( )

void Display_ar ( )

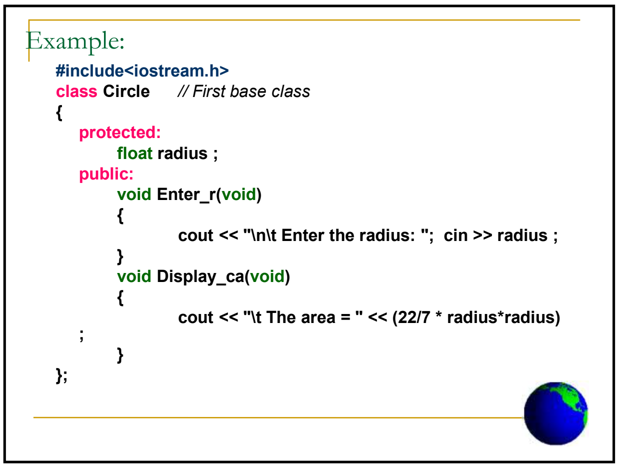
class Cylinder : public Circle, public Rectangle

public:

void volume_cy ( )

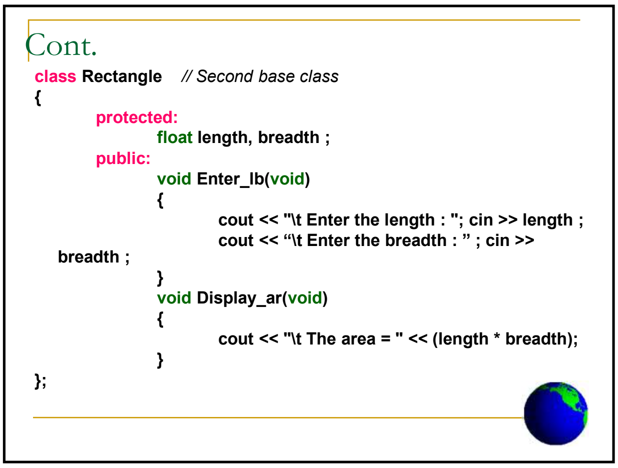
Cont.

- In the above figure**, Circle** and **Rectangle** are two **base classes** from which the **class Cylinder** is being inherited.
- The data members of both the base classes are declared in **protected** mode. Thus, the class **Cylinder** can access the data member **radius** of class **Circle** and data member **length, breadth** of the class **Rectangle**, but the **objects** of the class **Cylinder** cannot access these **protected data members**.
- The volume of the cylinder is equal to **22/7*(radius*radius*length).** Thus, instead of defining these data again, they can be inherited from the base classes **Circle** and **Rectangle** (**radius** from class **Circle** and **length** from class **Rectangle**).



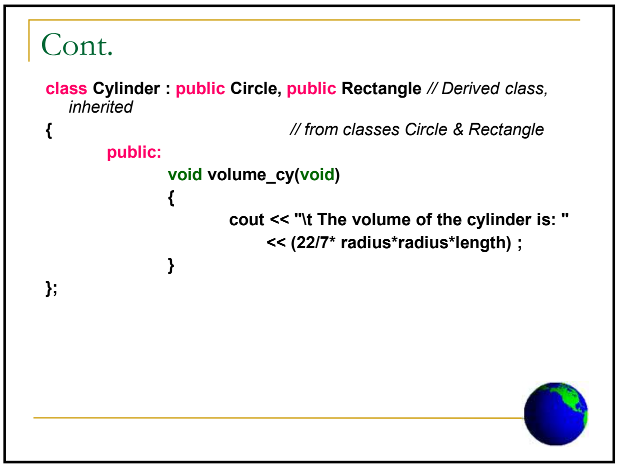
Example:

```
#include<iostream.h>
class Circle      // First base class
{
   protected:
        float radius ;
   public:
        void Enter_r(void)
        {
                cout << "\n\t Enter the radius: ";  cin >> radius ;
        }
        void Display_ca(void)
        {
                cout << "\t The area = " << (22/7 * radius*radius)
   ;
        }
};
```



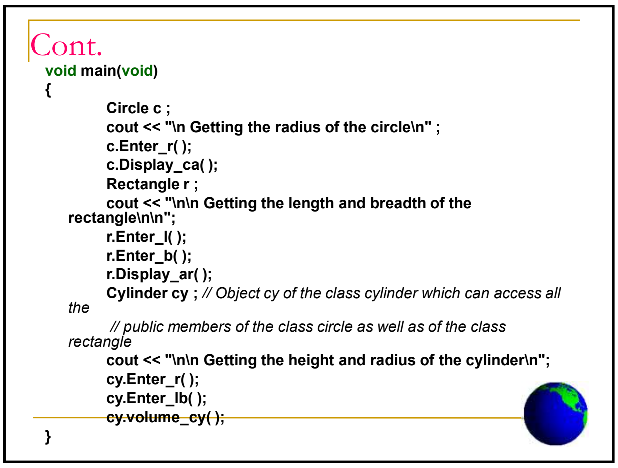
Cont.

```
class Rectangle    // Second base class
{
        protected:
                float length, breadth ;
        public:
                void Enter_lb(void)
                {
                        cout << "\t Enter the length : "; cin >> length ;
                        cout << "\t Enter the breadth : " ; cin >>
   breadth ;
                }
                void Display_ar(void)
                {
                        cout << "\t The area = " << (length * breadth);
                }
};
```



Cont.

```
class Cylinder : public Circle, public Rectangle // Derived class,
   inherited
{                                   // from classes Circle & Rectangle
        public:
                void volume_cy(void)
                {
                        cout << "\t The volume of the cylinder is: "
                             << (22/7* radius*radius*length) ;
                }
};
```



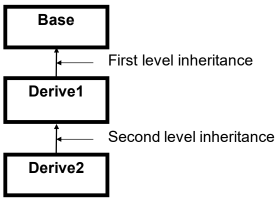
Cont.

```
void main(void)
{
        Circle c ;
        cout << "\n Getting the radius of the circle\n" ;
        c.Enter_r( );
        c.Display_ca( );
        Rectangle r ;
        cout << "\n\n Getting the length and breadth of the
   rectangle\n\n";
        r.Enter_l( );
        r.Enter_b( );
        r.Display_ar( );
        Cylinder cy ; // Object cy of the class cylinder which can access all
   the
         // public members of the class circle as well as of the class
   rectangle
        cout << "\n\n Getting the height and radius of the cylinder\n";
        cy.Enter_r( );
        cy.Enter_lb( );
        cy.volume_cy( );
}
```

Multilevel Inheritance:

- It has been discussed so far that a class can be derived from a class.
- **C++** also provides the facility of multilevel inheritance, according to which the derived class can also be derived by an another class, which in turn can further be inherited by another and so on.
- The following figure will illustrate the meaning of the multilevel inheritance.

Multilevel Inheritance



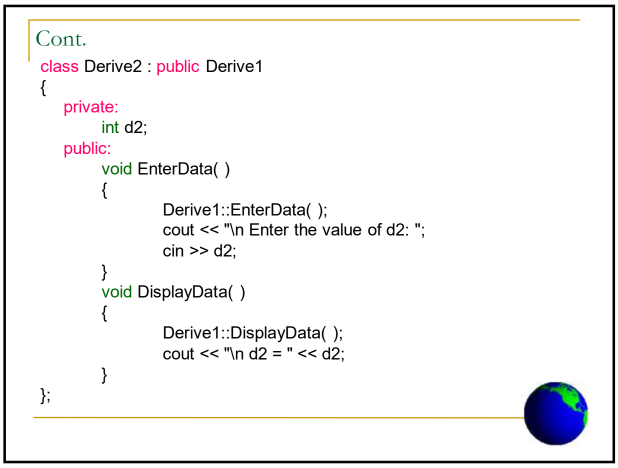
In the above figure, class **B** represents the base class. The class **D1** that is called **first level of inheritance,** inherits the class **B.** The derived class **D1** is further inherited by the class **D2,** which is called **second level of inheritance**.

Example: Multilevel Inheritance

```
#include<iostream.h>
class Base
{
    protected:
          int b;
    public:
          void EnterData( )
          {
                    cout << "\n Enter the value of b: ";
                    cin >> b;
          }
          void DisplayData( )
          {
                    cout << "\n b = " << b;
          }
};
```

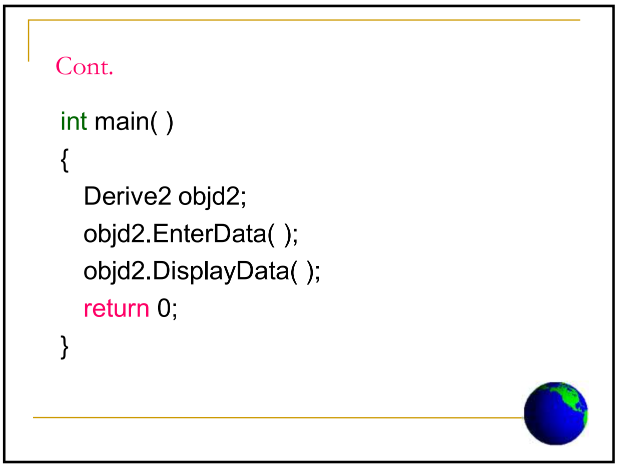
Cont.

```
class Derive1 : public Base
{
    protected:
          int d1;
    public:
          void EnterData( )
          {
                    Base:: EnterData( );
                    cout << "\n Enter the value of d1: ";
                    cin >> d1;
          }
          void DisplayData( )
          {
                    Base::DisplayData();
                    cout << "\n d1 = " << d1;
          }
};
```



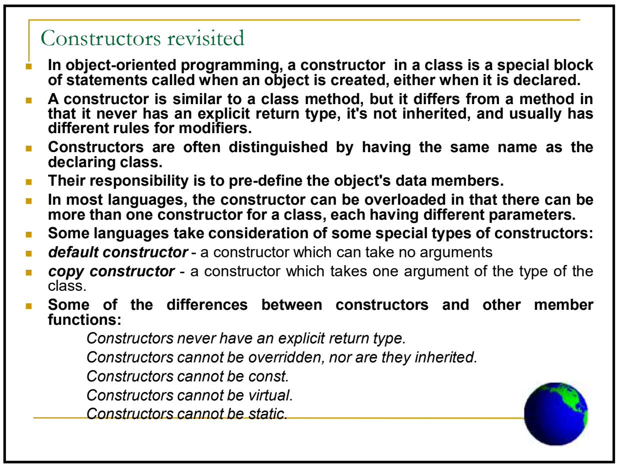
Cont.

```
class Derive2 : public Derive1
{
     private:
          int d2;
     public:
          void EnterData( )
          {
                    Derive1::EnterData( );
                    cout << "\n Enter the value of d2: ";
                    cin >> d2;
          }
          void DisplayData( )
          {
                    Derive1::DisplayData( );
                    cout << "\n d2 = " << d2;
          }
};
```



Cont.

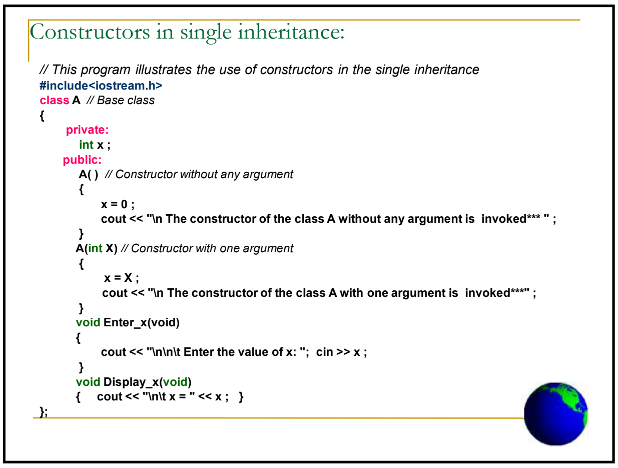
```
int main( )
{
   Derive2 objd2;
   objd2.EnterData( );
   objd2.DisplayData( );
   return 0;
}
```



Constructors revisited

- **In object-oriented programming, a constructor in a class is a special block of statements called when an object is created, either when it is declared.**
- **A constructor is similar to a class method, but it differs from a method in that it never has an explicit return type, it's not inherited, and usually has different rules for modifiers.**
- **Constructors are often distinguished by having the same name as the declaring class.**
- **Their responsibility is to pre-define the object's data members.**
- **In most languages, the constructor can be overloaded in that there can be more than one constructor for a class, each having different parameters.**
- **Some languages take consideration of some special types of constructors:**
- ***default constructor*** - a constructor which can take no arguments
- ***copy constructor*** - a constructor which takes one argument of the type of the class.
- **Some of the differences between constructors and other member functions:**

  Constructors never have an explicit return type.
  Constructors cannot be overridden, nor are they inherited.
  Constructors cannot be const.
  Constructors cannot be virtual.
  Constructors cannot be static.



Constructors in single inheritance:

```
// This program illustrates the use of constructors in the single inheritance
#include<iostream.h>
class A  // Base class
{
       private:
          int x ;
      public:
          A( )  // Constructor without any argument
          {
                x = 0 ;
                cout << "\n The constructor of the class A without any argument is  invoked*** " ;
          }
         A(int X) // Constructor with one argument
          {
                 x = X ;
                cout << "\n The constructor of the class A with one argument is  invoked***" ;
          }
         void Enter_x(void)
         {
                cout << "\n\n\t Enter the value of x: ";  cin >> x ;
          }
         void Display_x(void)
         {    cout << "\n\t x = " << x ;   }
};
```

Cont.

```
// ***************Derived  Class******************
class B : public A
{
        private:
              int y ;
         public:
              B( ) : A ( ) // Constructor of the derived class B without any argument
             {
                    y = 0 ;
                    cout << "\n The constructor of the class B without any argument is  invoked**" ;
              }
              // Constructor of the derived class B with two arguments
              B(int X,   // Argument for constructor A
                              int Y)   // Argument for constructor B
                                              : A(X)   // Call for the constructor A
              {
                    y = Y ;
                   cout << "\n The constructor of the class B with two arguments is invoked***" ;
              }
              void Enter_y(void)
              {     cout << "\t Enter the value of y: " ; cin >> y ;     }
              void Display_y(void)
              {     cout << "\n\t y = " << y ;    }
};
```

Cont.

```
     void main(void)
     {
                cout << "\n\n The first object b1 is in use********\n  " ;
                B b1 ;      // Invokes the constructor with  zero arguments
                b1.Enter_x( );
                b1.Enter_y( );
                b1.Display_x( );
                b1.Display_y( );
                cout << "\n\n The second object b2 is in use********\n  " ;
                B b2(5, 6);  // Invokes the constructor with two arguments
                b2.Display_x( );
                b2.Display_y( );
     }
```

Output:

```
The first object b1 is in use********

The constructor of class A without any argument is invoked*******

The constructor of class B without any argument is invoked*******

Enter the value of x: 10

Enter the value of y: 12

x = 10

y = 12

The second object b2 is in use********

The constructor of class A with one argument is invoked******

The constructor of class B with two arguments is invoked*****

x = 5

y = 6
```

Constructors in multilevel inheritance:

```
# include<iostream.h>
class A
{
      protected:
            int x ;
      public:
            A( )  // Constructor without argument
            {
                 x = 0 ;
                 cout << "\n Constructor of class A without any argument is invoked" ;
            }
            A(int X) // Constructor with one argument
            {
                  x = X ;
                  cout << "\n Constructor of class A with one argument is invoked" ;
            }
            void Enter_x(void)
            {  cout << "\n\t Enter the value of x: " ; cin >> x ; }
            void Display_x(void)
            { cout << "\n\t x = " << x ; }
};
```

Cont.

```
 class B : public A
 {
       protected:
              int y ;
       public:
              B( ) : A( )  // Constructor without argument
              {
                          y = 0;
                          cout << "\n Constructor of class B without any argument is invoked" ;
              }
                // Constructor with two arguments
              B( int X,  // Argument for constructor A
                          int Y )   // Argument for constructor B
                                   : A(X)   // Call for constructor A
              {
                          y = Y;
                          cout << "\n Constructor of class B with two arguments in invoked" ;
              }
              void Enter_y(void)
              { cout << "\t Enter the value of y: " ; cin >> y ;  }
              void Display_y(void)
              { cout << "\n\t y = " << y ;  }
 };
```

Cont.

```
class C : public B
{
      private:
              int z ;
      public:
              C( ) : B( )   // Constructor without argument
              {
                    z = 0;
                    cout << "\n Constructor of class C without any argument is invoked\n" ;
              }
              // Constructor with three arguments
              C(int X, int Y,   // Arguments for constructor B
                                  int Z)     // Argument for constructor C
                                            : B(X, Y)    // Call for constructor B
              {
                    z = Z ;
                    cout << "\n Constructor of class C with three arguments is invoked" ;
              }
              void Enter_z(void)
              { cout << "\t Enter the value of z: " ; cin >> z ; }
              void Display_z(void)
              { cout << "\n\t z = " << z ; }
};
```

Cont.

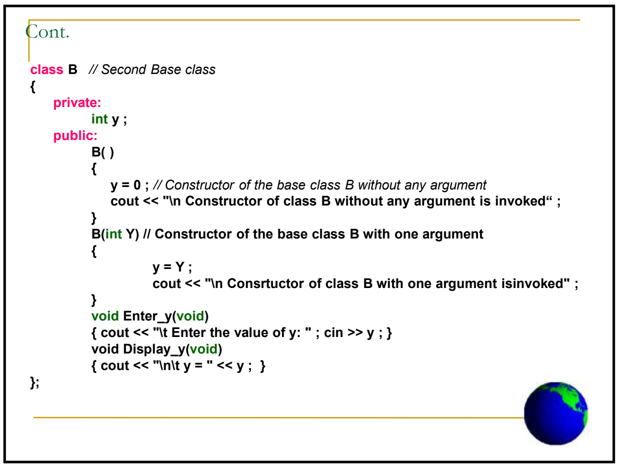
```
void main(void)
{
            cout << "\n The first object is in use now******** \n " ;
            C c1 ;
            c1.Enter_x( );
            c1.Enter_y( );
            c1.Enter_z( );
            c1.Display_x( );
            c1.Display_y( );
            c1.Display_z( );
            cout << "\n\n The second object is in use now******** \n" ;
            C c2(5, 6, 7) ;
            c2.Display_x( );
            c2.Display_y( );
            c2.Display_z( );
}
```

Output:

```
The first object is in use now*********

Constructor of class A without any argument is invoked
Constructor of class B without any argument is invoked
Constructor of class C without any argument is invoked

     Enter the value of x: 11
     Enter the value of y: 13
     Enter the value of z: 27

     x = 11
     y = 13
     z = 27
The second object is in use now*********

Constructor of class A with one argument is invoked
Constructor of class B with two arguments is invoked
Constructor of class C with three arguments is invoked

     x = 5
     y = 6
     z = 7
```

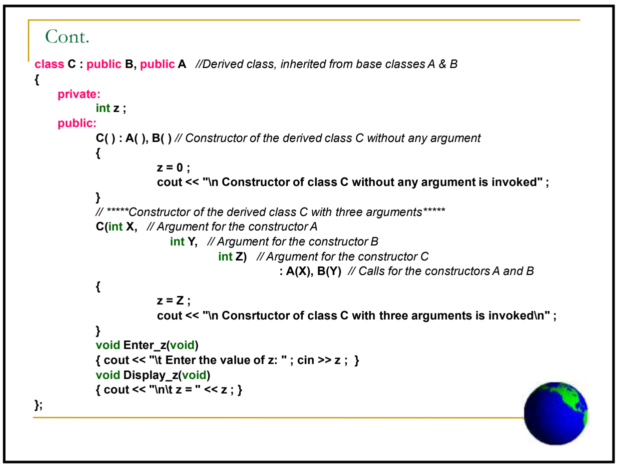
Constructors in multiple Inheritance:

```
#include<iostream.h>
class A  // First Base class
{
     private:
              int x ;
     public:
              A( )         // Constructor of the base class A without any argument
              {
                        x = 0 ;
                        cout << "\n Constructor of class A without any argument is invoked" ;
              }
              A(int X)   // Constructor of the base class A with one argument
              {
                        x = X ;
                       cout << "\n Constructor of class A with one argument is invoked" ;
              }

              void Enter_x(void)
              { cout << "\n\n\t Enter the value of x: " ; cin >> x ; }

              void Display_x(void)
              { cout << "\n\t x = " << x ;  }
};
```



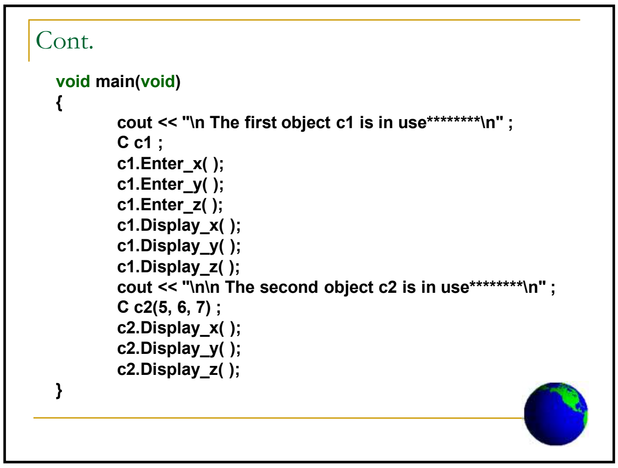
Cont.

```
class B   // Second Base class
{
     private:
               int y ;
     public:
               B( )
               {
                    y = 0 ; // Constructor of the base class B without any argument
                    cout << "\n Constructor of class B without any argument is invoked“ ;
               }
               B(int Y) // Constructor of the base class B with one argument
               {
                              y = Y ;
                              cout << "\n Consrtuctor of class B with one argument isinvoked" ;
               }
               void Enter_y(void)
               { cout << "\t Enter the value of y: " ; cin >> y ; }
               void Display_y(void)
               { cout << "\n\t y = " << y ;  }
};
```



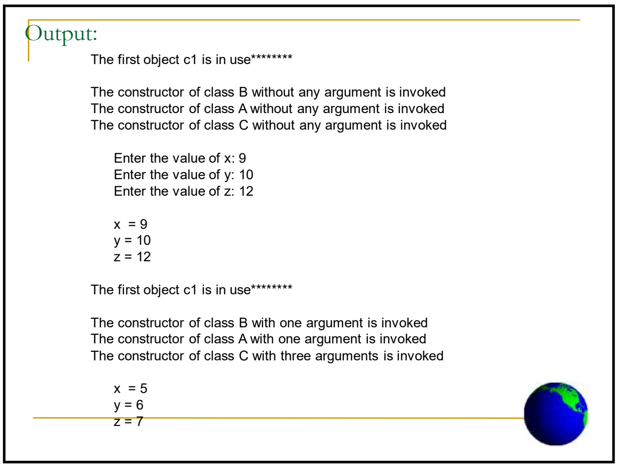
Cont.

```
class C : public B, public A   //Derived class, inherited from base classes A & B
{
     private:
               int z ;
     public:
               C( ) : A( ), B( ) // Constructor of the derived class C without any argument
               {
                              z = 0 ;
                              cout << "\n Constructor of class C without any argument is invoked" ;
               }
               // *****Constructor of the derived class C with three arguments*****
               C(int X,   // Argument for the constructor A
                                   int Y,   // Argument for the constructor B
                                             int Z)   // Argument for the constructor C
                                                            : A(X), B(Y)  // Calls for the constructors A and B
               {
                              z = Z ;
                              cout << "\n Consrtuctor of class C with three arguments is invoked\n" ;
               }
               void Enter_z(void)
               { cout << "\t Enter the value of z: " ; cin >> z ;  }
               void Display_z(void)
               { cout << "\n\t z = " << z ; }
};
```



Cont.

```
  void main(void)
  {
            cout << "\n The first object c1 is in use********\n" ;
            C c1 ;
            c1.Enter_x( );
            c1.Enter_y( );
            c1.Enter_z( );
            c1.Display_x( );
            c1.Display_y( );
            c1.Display_z( );
            cout << "\n\n The second object c2 is in use********\n" ;
            C c2(5, 6, 7) ;
            c2.Display_x( );
            c2.Display_y( );
            c2.Display_z( );
  }
```



Output:

```
The first object c1 is in use********

The constructor of class B without any argument is invoked
The constructor of class A without any argument is invoked
The constructor of class C without any argument is invoked

     Enter the value of x: 9
     Enter the value of y: 10
     Enter the value of z: 12

     x  = 9
     y = 10
     z = 12

The first object c1 is in use********

The constructor of class B with one argument is invoked
The constructor of class A with one argument is invoked
The constructor of class C with three arguments is invoked

     x  = 5
     y = 6
     z = 7
```