Learners Support Publications
www.lsp4you.com

Inheritance: Extending Classes

Introduction

- Reusability is an important feature of OOP.
- C++ strongly supports the concept of reusability.

Introduction

continue …

- The mechanism of deriving a new class from an old one is called inheritance (or derivation).
- The old class is referred to as base class.
- The new class is called the derived class or subclass.

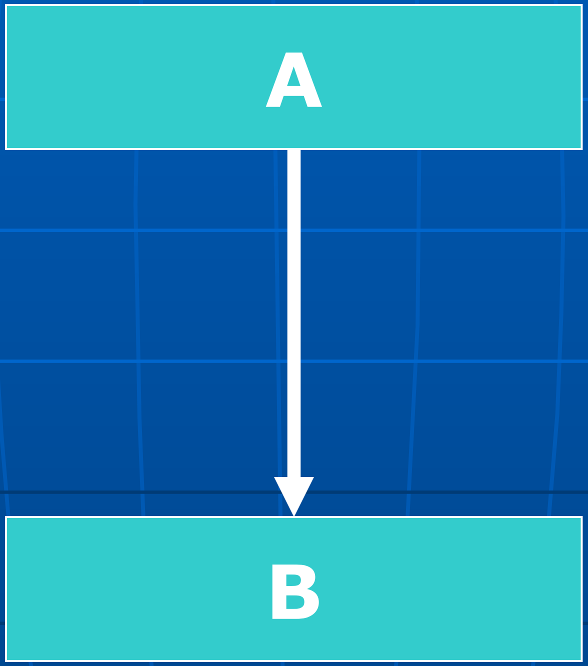
Introduction

continue …

- The derived class inherits some or all of the traits from the base class.
- A class can also inherit properties from more than one class or from more than one level.

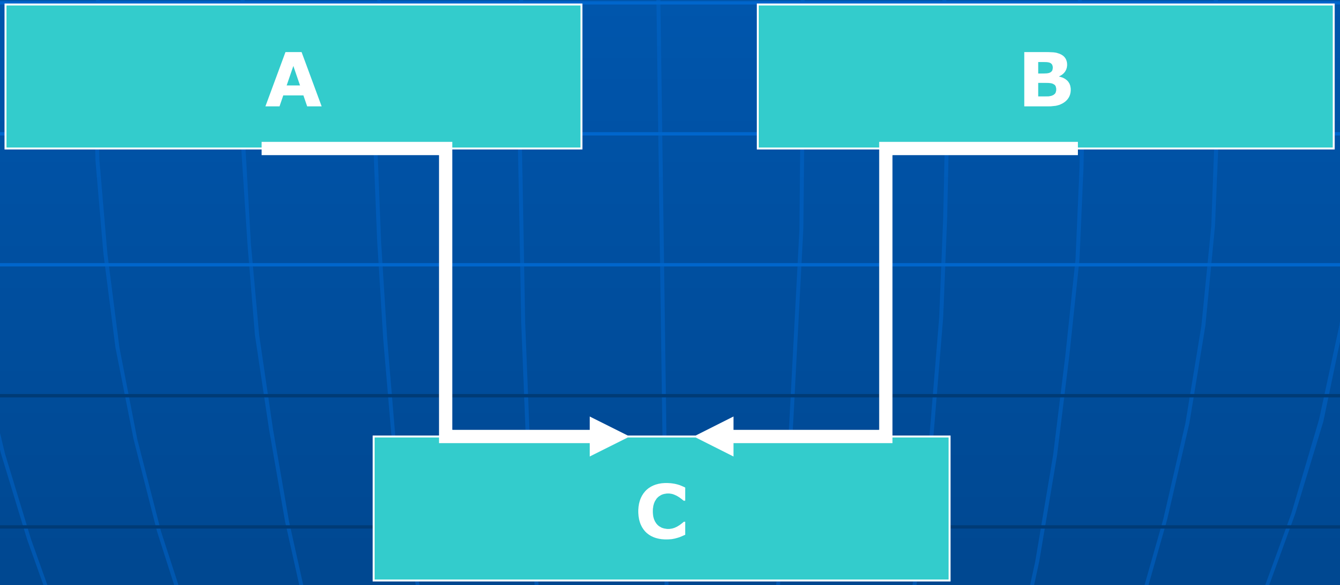
Single Inheritance

- A derived class with only one base class.



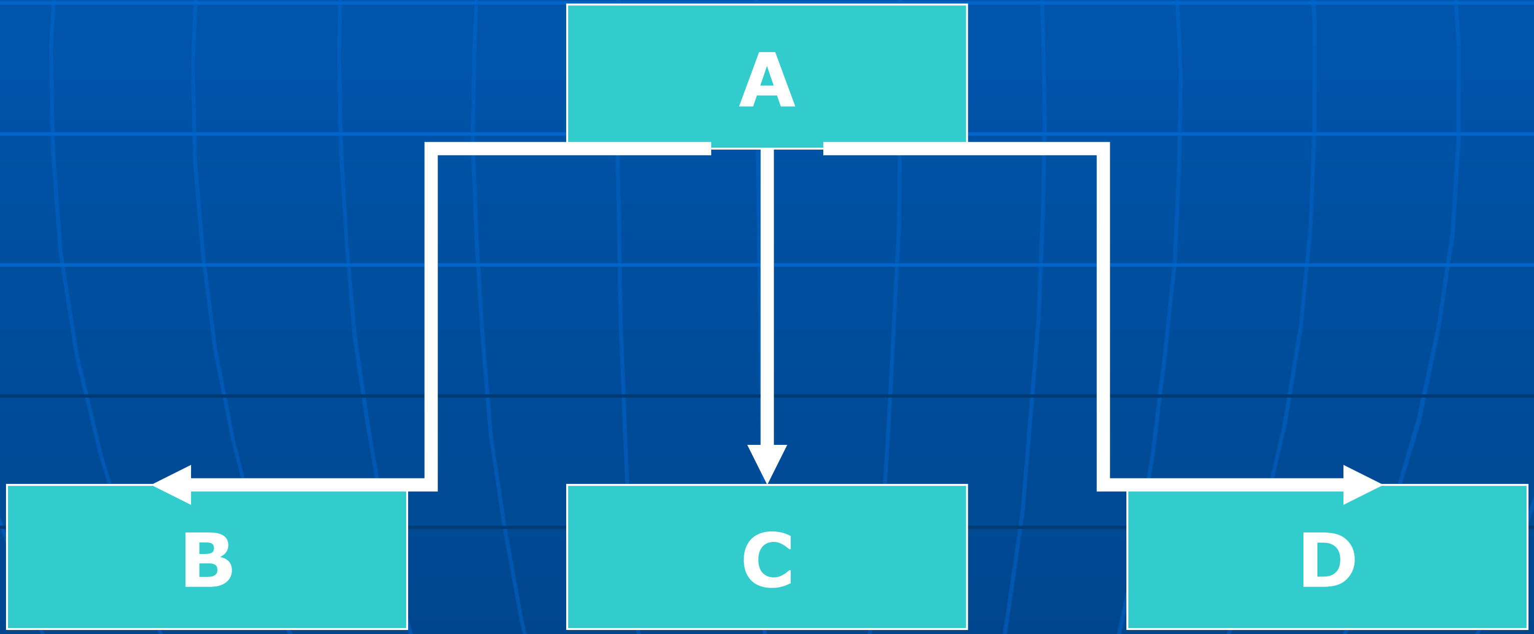
Multiple Inheritance

- A derived class with several base classes.



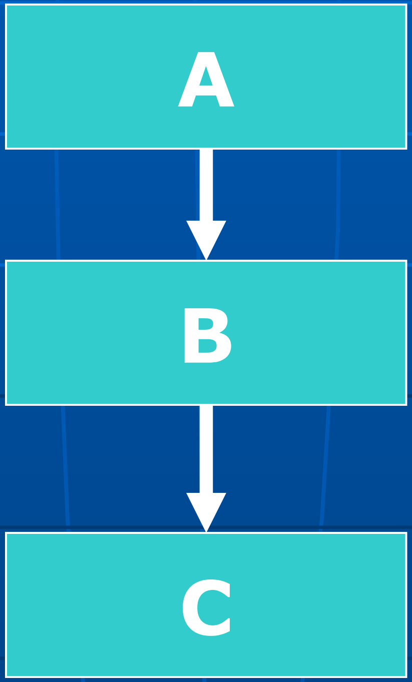
Hierarchical Inheritance

- A traits of one class may be inherited by more than one class.



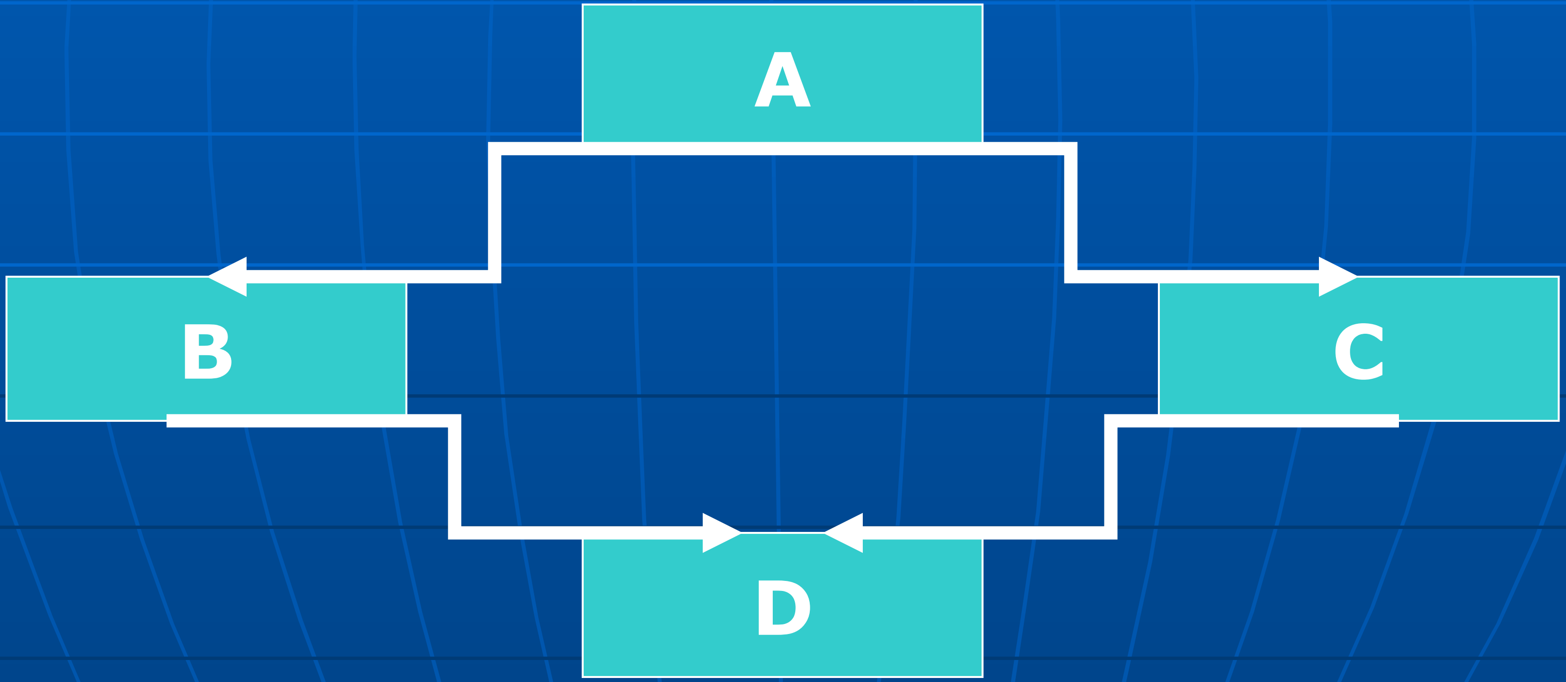
Multilevel Inheritance

- The mechanism of deriving a class from another derived class.



Hybrid Inheritance

- The mechanism of deriving a class by using a mixture of different methods.



Derived Classes

- A derived class can be defined by specifying its relationship with the base class in addition to its own details.

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

rive Classes

continue …

```
class derived-class-na            bility-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 is optional and , if present, may be either **private** or **public**.

The default visibility mode is **private**.

Visibility mode specifies whether the features of the base class are **derived privately or publicly**.

Derived Classes

continue …

- When a base class is privately derived by a derived class, “public members” of the base class become “private members” of the derived class.
- Therefore the members of the derived class can only access the public members of the base class.
- They are inaccessible to the objects of the derived class.
- No member of the base class is accessible to the objects of the derived class.

Derived Classes

continue …

- When a base class is publicly inherited, ”public members” of the base class become the “public members” of the derived class.

- They are accessible to the objects of the derived class.

Derived Classes

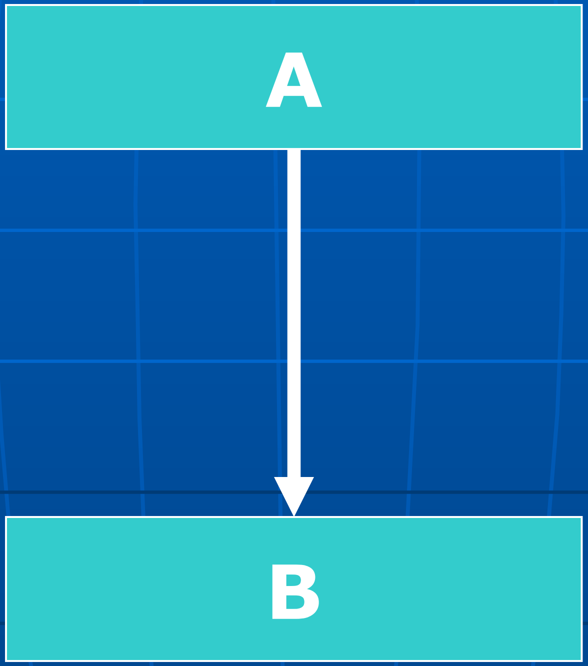
continue …

- The private members of the base class are not inherited in both the cases (publicly/privately inherited).
- The private members of a base class will never become the members of its derived class.

Inheritance

- In inheritance, some of the base class data elements and member functions are inherited into the derived class.
- We can add our own data and member functions for extending the functionality of the base class.
- It is a powerful tool for incremental program development.
- Can increase the capabilities of an existing class without modifying it.

Single Inheritance



Making a Private Member Inheritable

- By making the visibility limit of the private members to the public.
- The visibility modifier "protected" can be used for this purpose.
- A member declared as "protected" is accessible by the member functions within its class and any class immediately derived from it.
- It can not be accessed by the functions outside these two classes.

Making a Private Member Inheritable

continue …

```
class alpha
{
            private : // optional
                        .........       // visible to the member within
its class
                        .........
            protected :
                        .........   // visible to member functions
                        .........   // of its own and immediate derived
class
            public :
                        .........   // visible to all functions
                        .........   // in the program
};
```

Protected Member

continue …

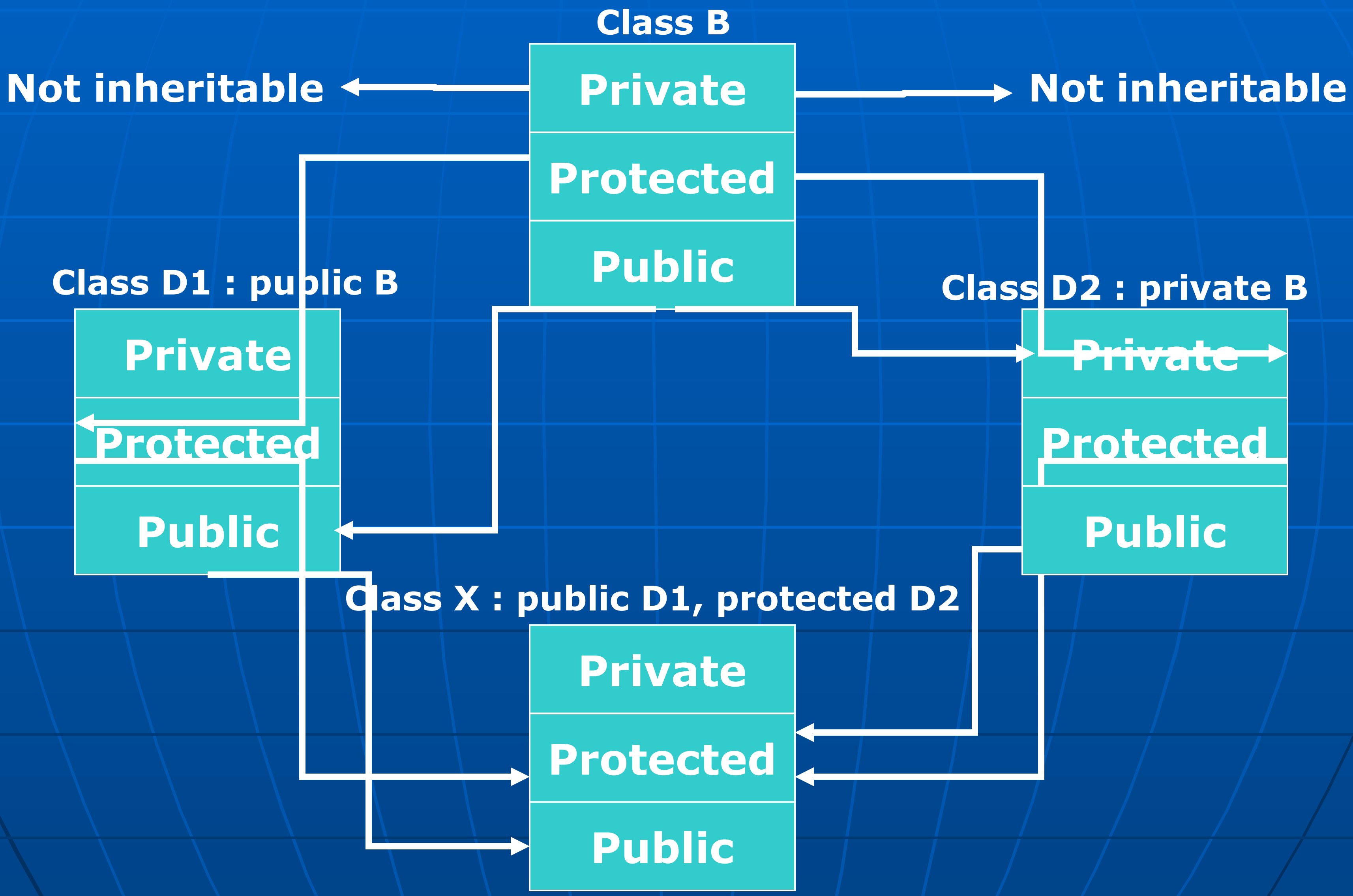
- When a protected member is inherited in public mode, it becomes protected in the derived class.
- They are accessible by the member functions of the derived class.
- And they are ready for further inheritance.

- When a protected member is inherited in private mode, it becomes private in the derived class.
- They are accessible by the member functions of the derived class.
- But, they are not available for further inheritance.

Protected Derivation

- It is possible to inherit a base class in protected mode – protected derivation.
- In protected derivation, both the public and protected members of the base class become protected members of the derived class.

Effect of Inheritance on the visibility of Members



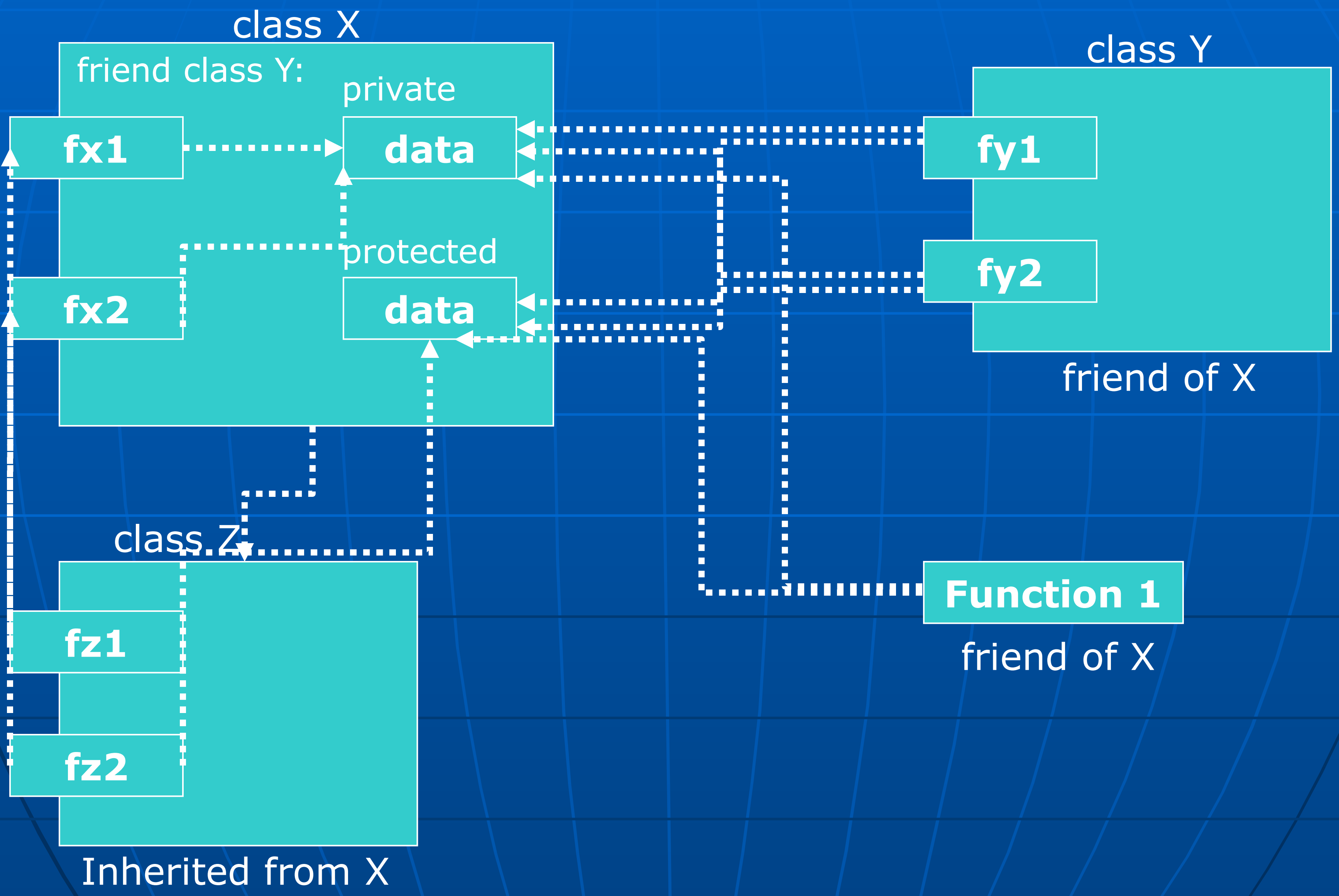
Visibility

Base class visibility	Derived class visibility		
	Public Derivation	Private Derivation	Protected Derivation
Private →	Not Inherited	Not Inherited	Not Inherited
Protected →	Protected	Private	Protected
Public →	Public	Private	Protected

Access Control to Data Members

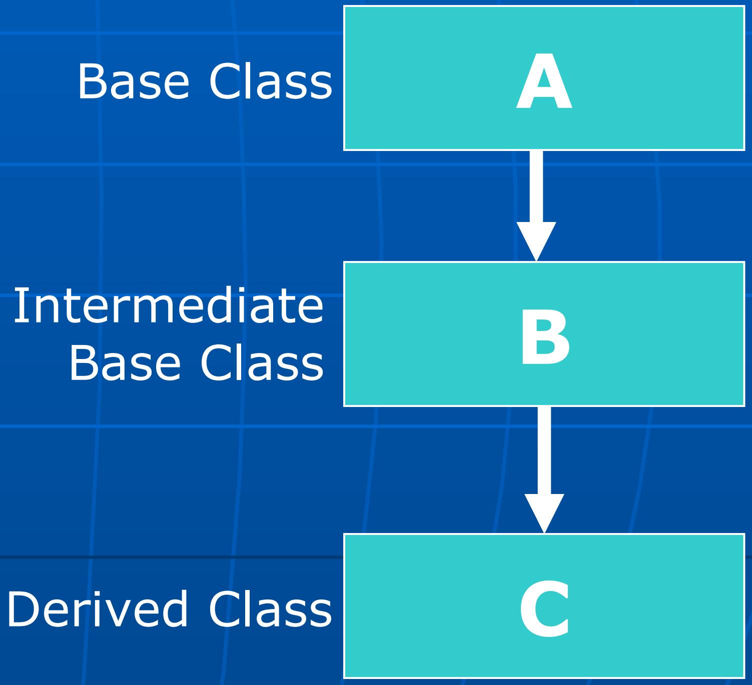
- Functions that can have access to the private and protected members of a class:
  - A function that is a friend of the class.
  - A member function of a class that is a friend of the class.
  - A member function of a derived class.

Access mechanism in classes



Multilevel Inheritance

- The class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.
- Class B provides a link for the inheritance between A and C.
- The chain ABC is known as inheritance path.



Multilevel Inheritance

continue …

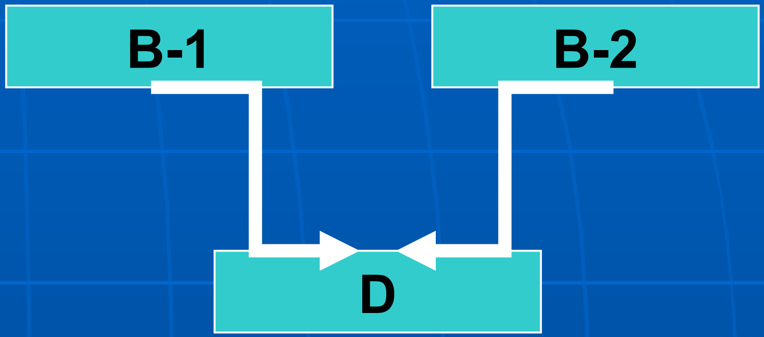
```
class A { ………} ;                    // Base Class

class B : public A { ……… } ;  // B derived from A

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

Multiple Inheritance

- A class can inherit the attributes of two or more classes.
- Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.



- It is like a child inheriting the physical features of one parent and the intelligence of another.

Multiple Inheritance

continue …

```
class D: visibility B-1, visibility B-2, ……
{
        ………
        ……… (Body of D)
        ………
};
```

- Where, visibility may be either public or private.
- The base classes are separated by comma.

Ambiguity Resolution in Inheritance

```
class M
{
   public:
         void display (void)
        { cout << "Class M \n";}
};
class N
{
   public:
         void display (void)
        { cout << "Class N \n";}
};
```

```
class P : public M, public N
{
   public:
         void display (void)
        { M :: display();
           //here ambiguity can be //resolved}
};
void main( )
{
     P p;
     p.display( );
}
```

In Multiple Inheritance

Ambiguity Resolution in Inheritance

continue …

```
class A
{
   public:
   void display (void)
      { cout << "Class A \n";}
};
class B : public A
{
   public:
   void display (void)
      { cout << "Class B \n";}
};
```

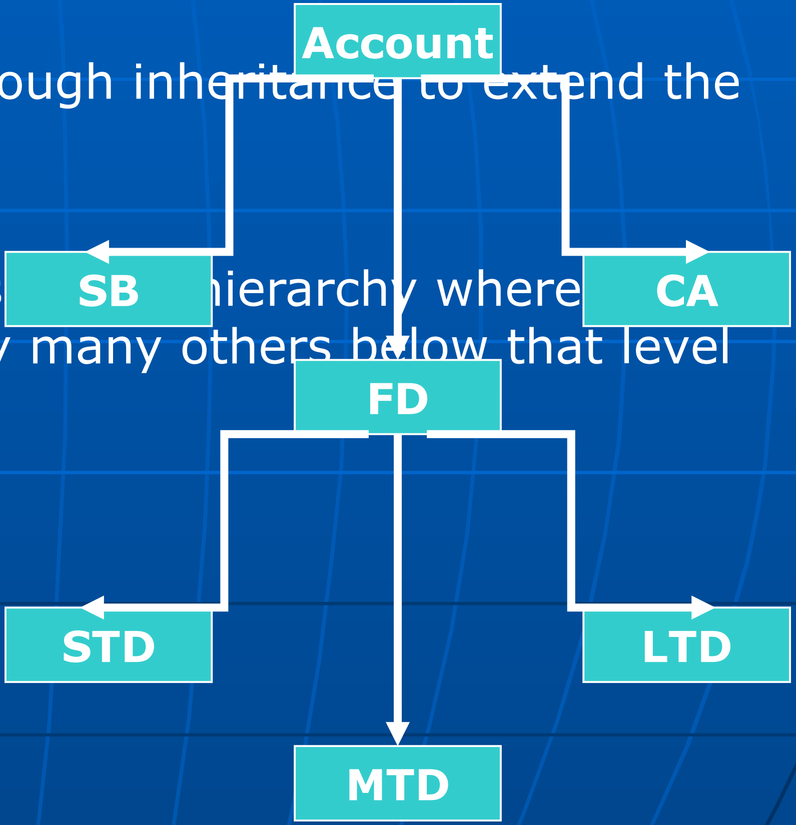
```
void main( )
{
   B b;
   b.display( ); // in B
   b.A::display( ); // in A
   b.B::display( ); // in B
}
```

Ambiguity can be resolved by specifying the function with class name and scope resolution operator to invoke.

In Single Inheritance

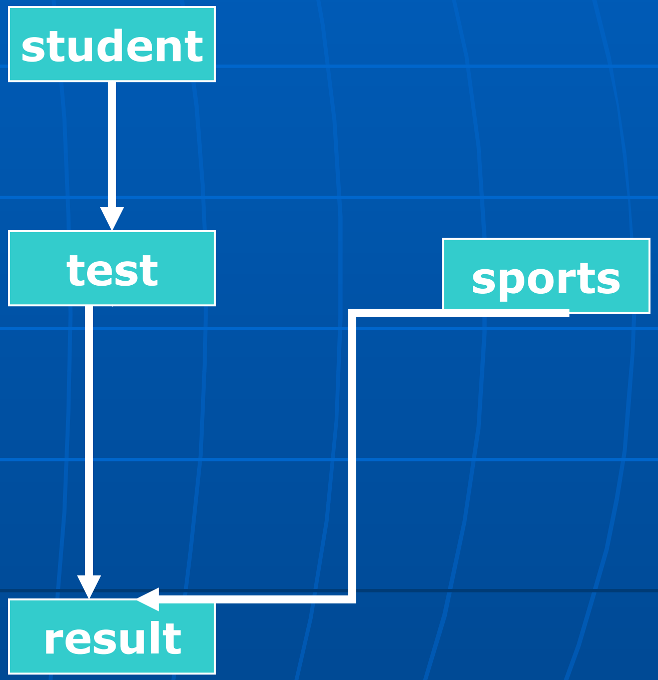
Hierarchical Inheritance

- Inheritance support hierarchical design of a program.
- Additional members are added through inheritance to extend the capabilities of a class.
- Programming problems can be cas [illegible] hierarchy where features of one level are shared by many others below that level



Hybrid Inheritance

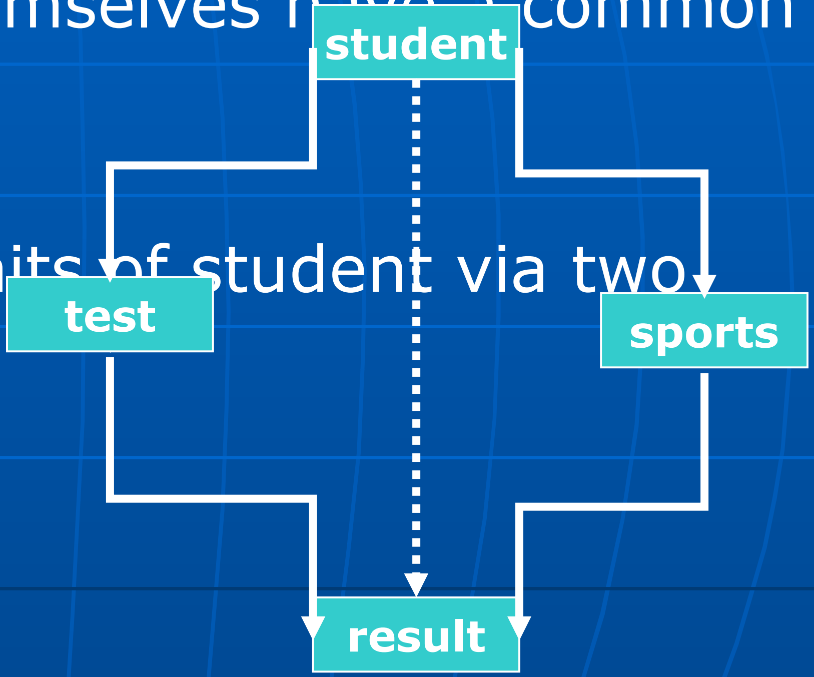
- Applying Two or more types of inheritance together.



Virtual Base Classes

- Here the result class has two direct base classes test and sports which themselves have a common base class student.

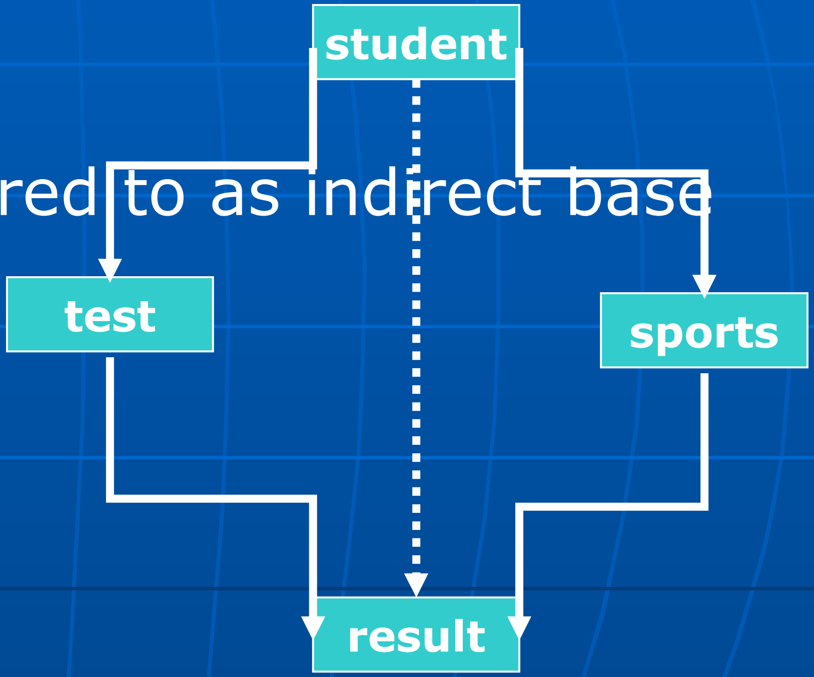
- The result inherits the traits of student via two separate paths.



Virtual Base Classes

continue …

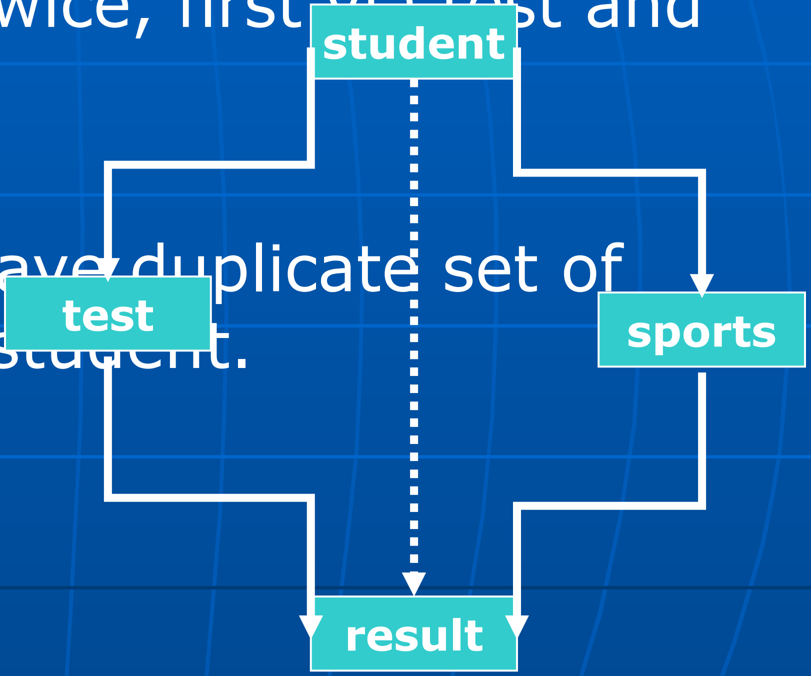
- It can also inherit directly as shown by the broken line.
- The student class is referred to as indirect base class.



Virtual Base Classes

continue …

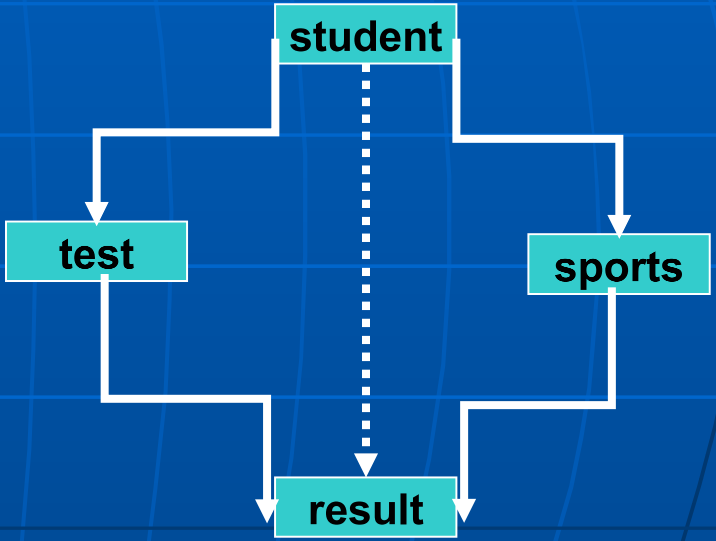
- All the public and protected members of student are inherited into result twice, first via test and again via sports.
- This means result class have duplicate set of members inherited from student.



Virtual Base Classes

continue …

```
class student
{
    ………
};
class test : virtual public student
{
    ………
};
class sports : public virtual student
{
    ………
};
class result : public test, public sports
{
    ………\
};
```


$$\begin{tikzcd}
 & \text{student} \arrow[dl] \arrow[dr] \arrow[d, dashed] & \\
\text{test} \arrow[r] & \text{result} & \text{sports} \arrow[l]
\end{tikzcd}$$

Abstract Classes

- An abstract class is one that is not used to create objects.
- An abstract class is designed only to act as a base class.
- It is a design concept in program development and provides a base upon which other classes may be built.

Constructors in Derived Classes

- If no base class constructor takes any arguments, the derived class need not have a constructor function.

- If any base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructors.

Constructors in Derived Classes

continue …

- When both the derived and base class contain constructors, the base constructor is executed first and then the constructor in the derived class is executed.

- In case of multiple inheritance, the base class constructors are executed in the order in which they appear in the declaration of the derived class.

Constructors in Derived Classes

continue …

- In a multilevel inheritance, the constructors will be executed in the order of inheritance.

- Since the derived class takes the responsibility of supplying initial values to its base classes, we supply the initial values that are required by all the classes together, when a derived class object is declared.

Constructors in Derived Classes

continue …

- 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.

- The base constructors are called and executed before executing the statements in the body of the derived constructor.

Constructors in Derived Classes

continue …

- The header line of *derived-constructor* function contains two parts separated by a colon (:).
  - The first part provides the declaration of the arguments that are passed to the derived constructor.
  - The second part lists the function calls to the base constructors.

Defining Derived Constructors

continue …

```
Derived-constructor(Arglist1, Arglist2, … ArglistN,
     ArglistD) :
base1(arglist1),
base2(arglist2),
…
baseN(arglistN)
{

}
```

Member Classes : Nesting of Classes

- Inheritance is the mechanism of deriving certain properties of one class into another.
- C++ supports a new way of inheriting classes:
  - An object can be collection of many other objects.
  - A class can contain objects of other classes as its members.

Member Classes : Nesting of Classes

continue …

```
class alpha { ……… };
class beta { ……… };
class gamma
{
    alpha a;   // an object of class alpha
    beta b; // an object of class beta
    ………
};
```

Member Classes : Nesting of Classes

continue …

```
class alpha { ……… };
class beta { ……… };
class gamma
{
      alpha a;
      beta b;
      ………
};
```

- All objects of gamma class will contain the objects a and b.
- This is called *containership* or *nesting*.

Member Classes : Nesting of Classes

continue …

- An independent object is created by its constructor when it is declared with arguments.
- A nested object is created in two stages:
  - The member objects are created using their respective constructors.
  - Then ordinary members are created.

- Constructors of all the member objects should be called before its own constructor body is executed.

Member Classes : Nesting of Classes

continue …

```
class gamma
{
        ………
        alpha a;
        beta b;
    public:
        gamma(arglist): alpha(arglist1), beta(arglist2)
         {     body of the constructor     }
};
```