CE142: OBJECT ORIENTED PROGRAMMING WITH C++

December – April 2019

Unit– 09

Inheritance

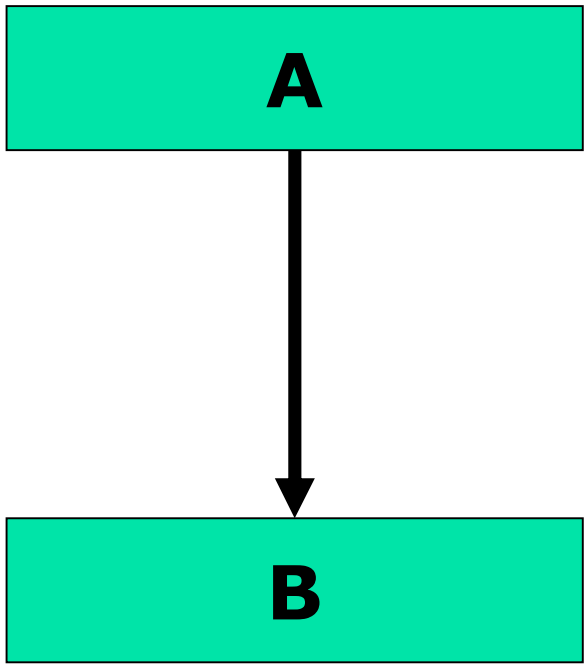
Devang Patel Institute of Advance Technology and Research

Introduction

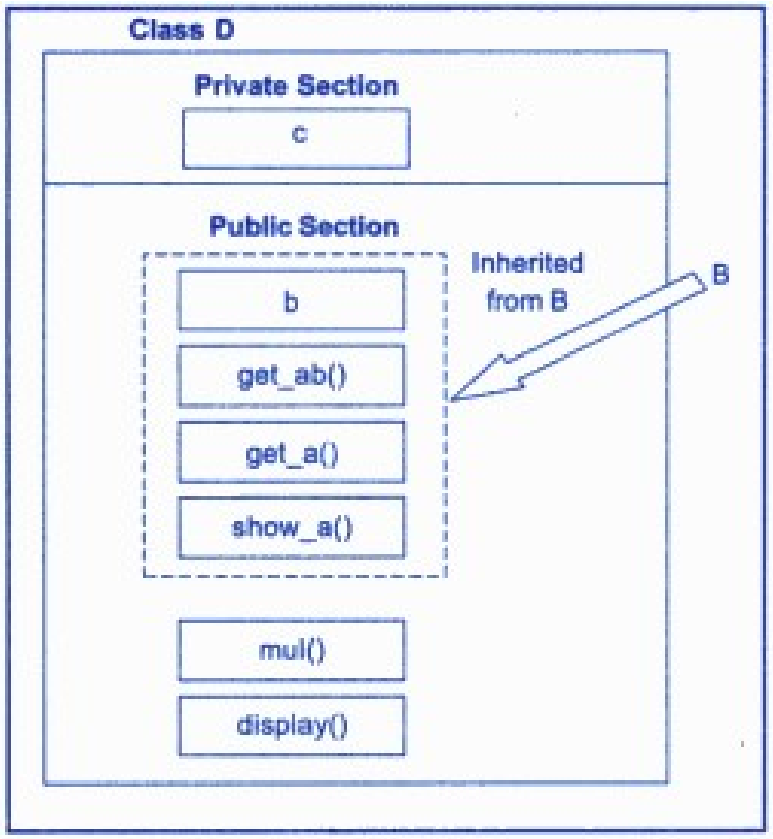
- Reusability is an important feature of OOP.
  - The mechanism of deriving a new class from an old one is called inheritance (or derivation).
  - The old class is referred to as base class.
  - The new class is called the derived class or subclass
- The derived class inherits some or all of the traits 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.

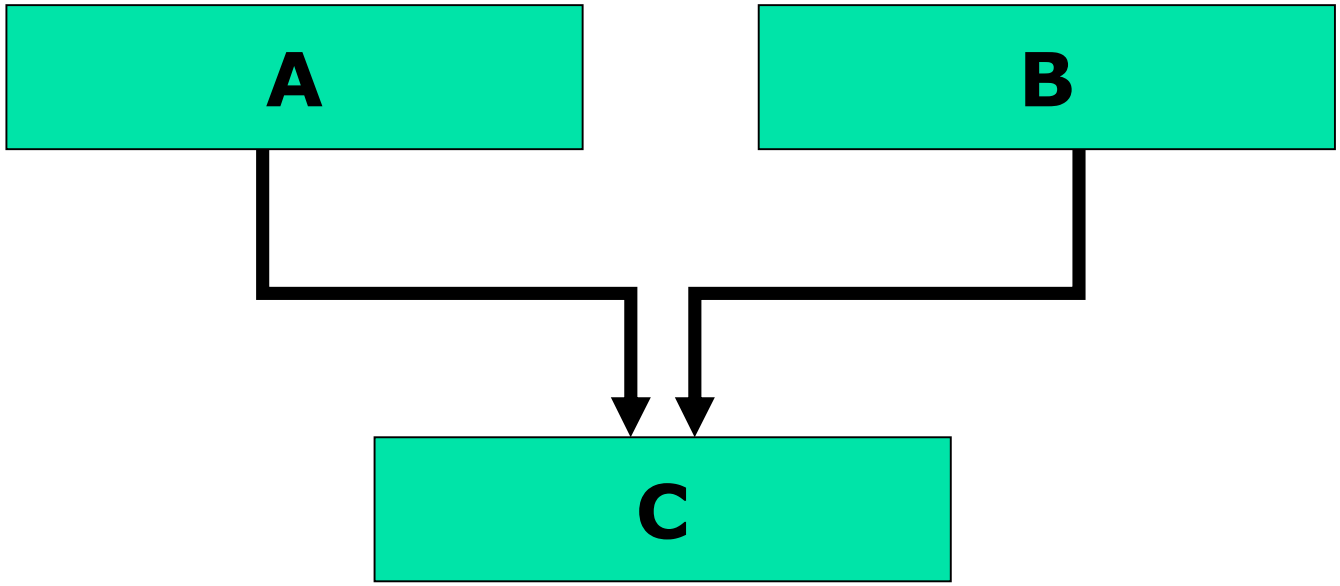


Single Inheritance



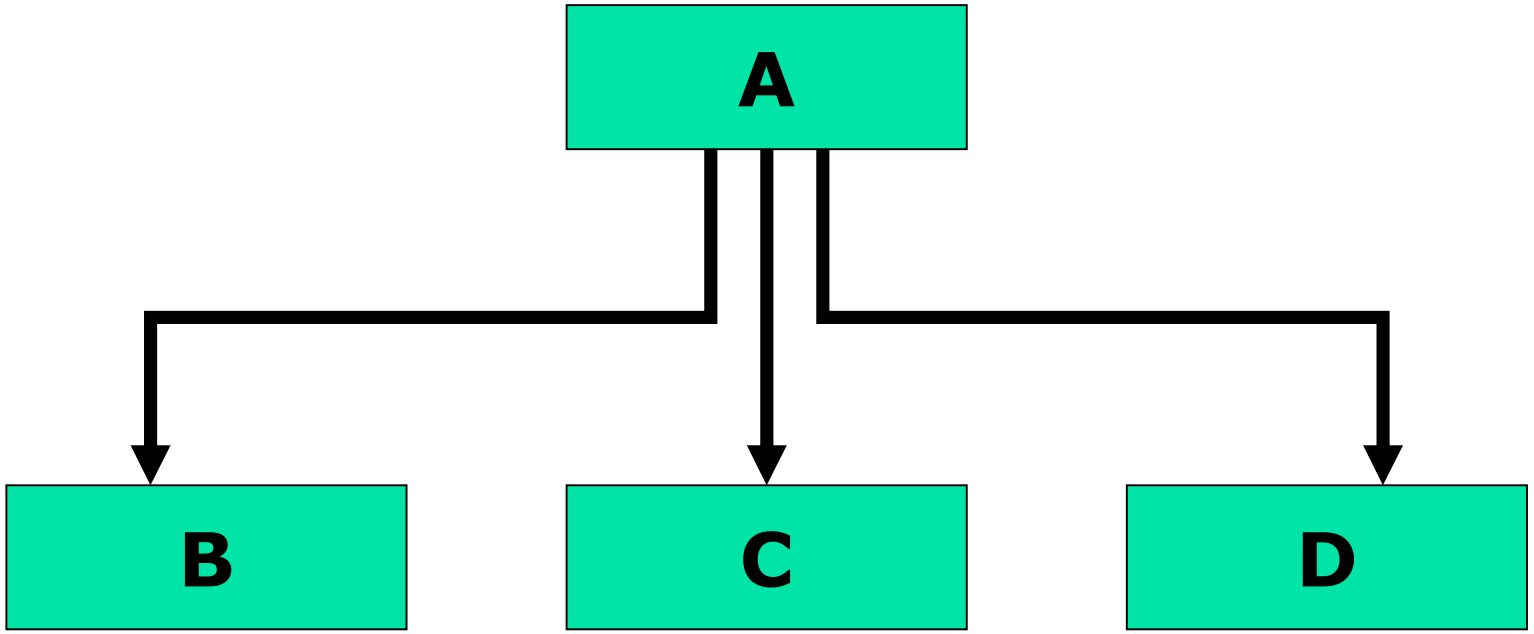
Multiple Inheritance

- A derived class with several base classes.



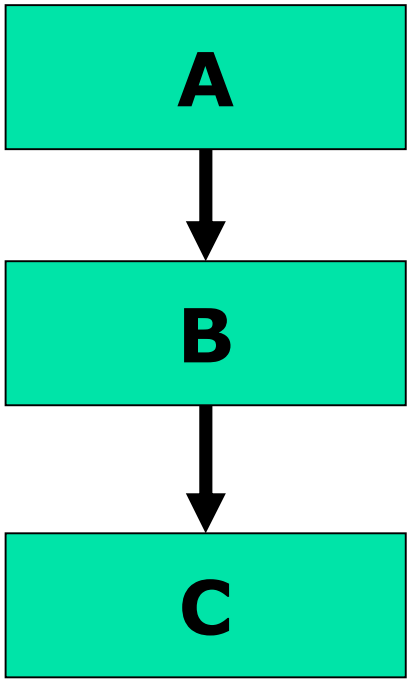
Hierarchical Inheritance

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



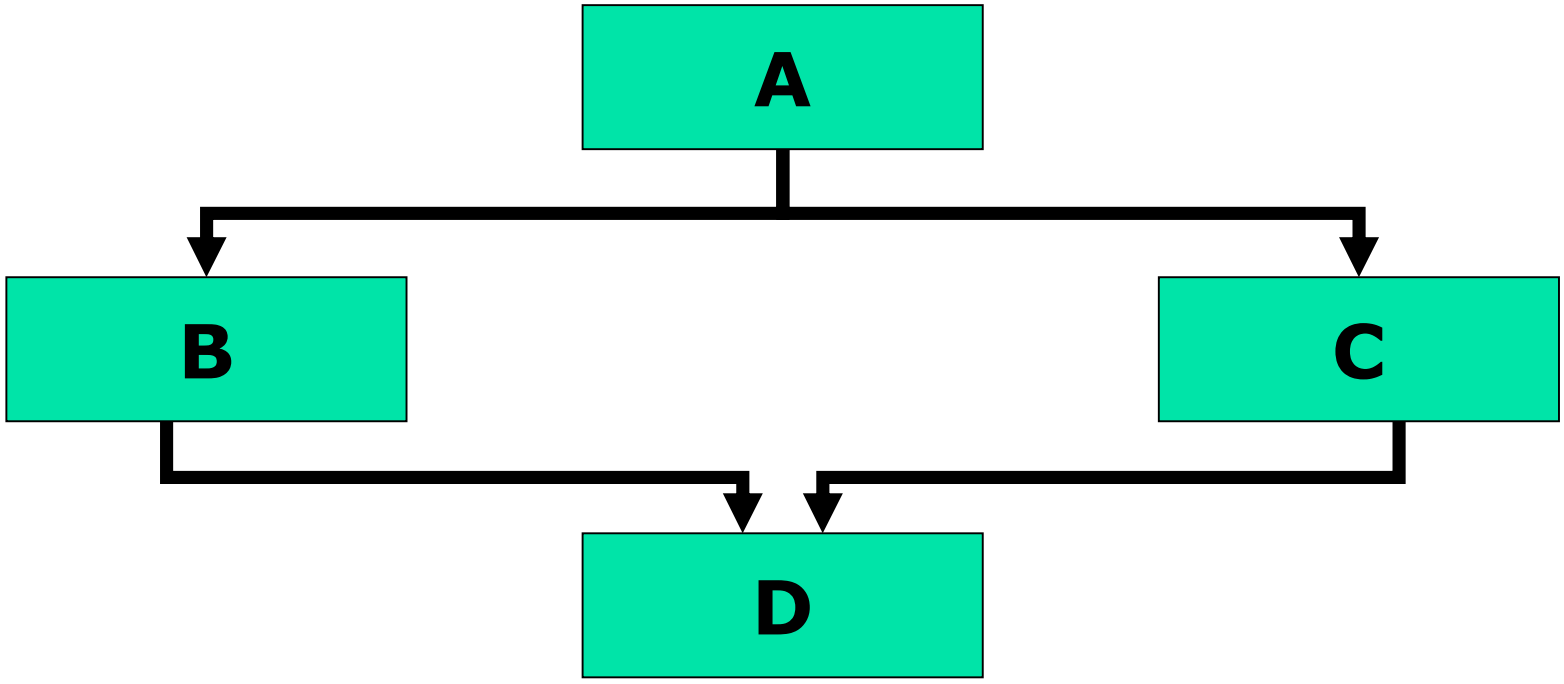
Multilevel Inheritance

- 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

- Visibility mode is either private or public or protected
  - By default its private
  - It specifies whether the features of the base class are privately derived or publicly derived

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

Derived Classes

```
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 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**.

Prg-1

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

class A
{
    public:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};

int main()
{
    B obj;
    obj.display();
    return 0;
}
```

O/P

Base class content

Prg-2

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

class A
{
    private:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};

int main()
{
    B obj;
    obj.display();
    return 0;
}
```

O/P

void A::display()' is private

Prg-3

```
#include <iostream>
using namespace std;
#include <stdio.h>
class A
{
    public:
       int i;
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};
```

```
int main()
{
    B obj;
    obj.display();
    obj.i=10;
    printf("%d",obj.i);
    return 0;
}
```

O/P

Base class content. 10

Prg-4

```
#include <iostream>
using namespace std;
#include <stdio.h>
class A
{
    private:
        int i;
    public:
      void display()
      {
          cout<<"Base class content";
      }
};
class B : public A
{
public:
        seti()
        {
            i=10;
        }
};
int main()
{
    B obj;
    obj.seti();
    obj.display();
return 0;
}
```

O/P

Error: 'int A::I is private'

Prg-5

```
#include <iostream>
using namespace std;
#include <stdio.h>
class A
{
    private:
        int i;
    public:
      void display()
      {
          cout<<"Base class content Value.“<<i;
      }
      int seti()
         {
            i=10;
         }
};
class B : public A
{
};
```

```
iint main()
{
    B obj;
    obj.seti();
    obj.display();
return 0;
}
```

O/P

Base class content Value: 10

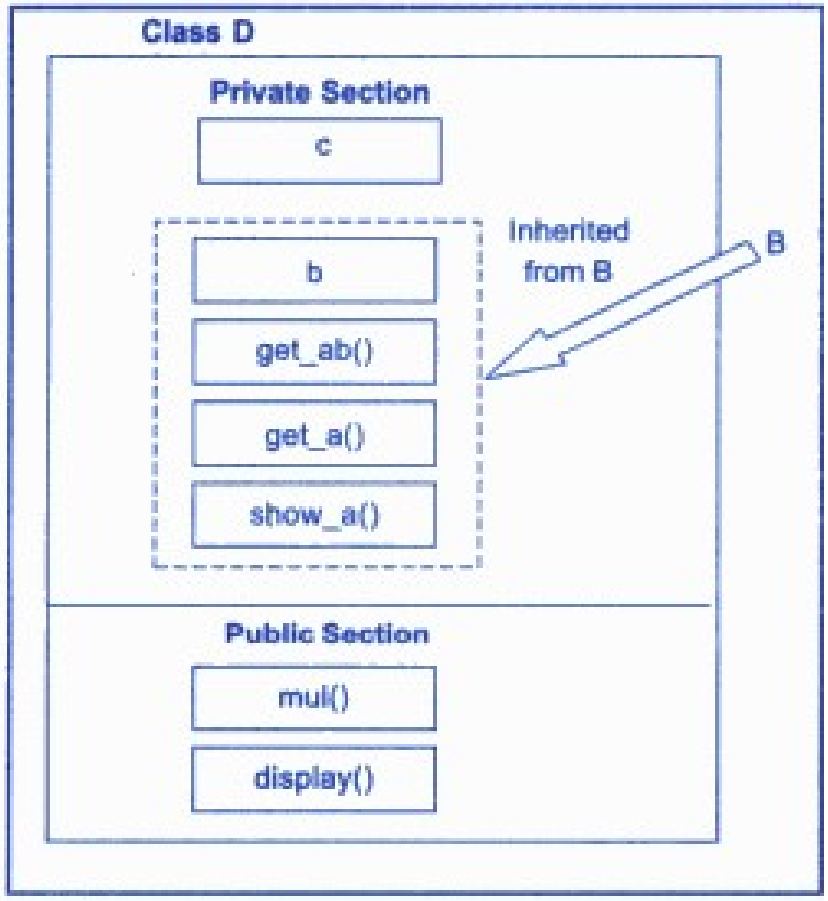
Derived Classes: Public Visibility

- 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.
- 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.

Derived Classes : Private Visibility

- 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 : Private Visibility



Prg-1

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

class A
{
    public:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : private A
{

};

int main()
{
    B obj;
    obj.display();
    return 0;
}
```

O/P

Base class content

Error

Prg-1

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

class A
{
    public:

      void display()
      {
          cout<<"Base class content.";
      }
};

class B : private A
{
public:
    void displayderive()
    {
        display();
    }
};

int main()
{
    B obj;
    obj.displayderive();
    return 0;
}
```

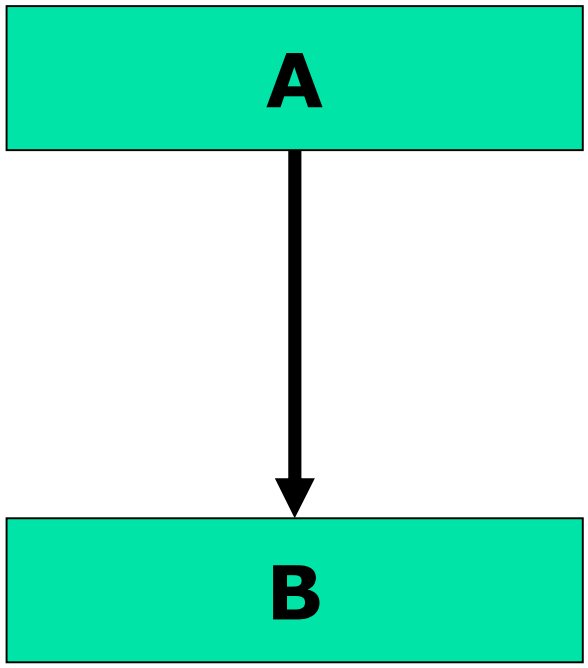
O/P

Base class content

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



```
#include <iostream>
using namespace std;
class base    //single base class
{
   public:
     int x;
   void getdata()
   {
     cout << "Enter the value of x = "; cin >> x;
   }
 };
class derive : public base    //single derived class
{
   private:
    int y;
   public:
   void readdata()
   {
     cout << "Enter the value of y = "; cin >> y;
   }
   void product()
   {
     cout << "Product = " << x * y;
   }
 };
```

```
int main()
{
    derive a;       //object of derived class
    a.getdata();
    a.readdata();
    a.product();
    return 0;
}           //end of program
```

Output:

```
Enter the value of x = 12
Enter the value of y = 13
Product = 156
```

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
};
```

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

class worker
{
  int age;
  char name [10];
  public:
    void get()
    {
      cout << "Enter your Name : ";
      cin >> name;

      cout << "Enter your Age : ";
      cin >> age;
    }
    void show()
    {
      cout << "\nYour name is " << name << "\nYour age is " << age;
    }
};
```

```
class manager : private worker
//derived class inherit the base class using private derivation
{
  int now;
  public:
    void get()
    {
      worker::get();
      //calling base class get function
      cout << "Enter the number of workers under you : ";
      cin >> now;
    }
    void show()
    {
      worker::show();
      cout << "\nNumber of workers under you is " << now;
    }
};

int main ()
{
  manager m1;
  m1.get();
  m1.show();
  return 0;
}
```

Output:

```
Enter your Name : DEV
Enter your Age : 23

Enter the number of workers under you : 14

Your name is DEV
Your age is 23
Number of workers under you is 14
```

Protected Member

- 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.

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

class worker
{
  int age;
  char name [10];
  public:
    void get()
    {
      cout << "Enter a name:";
      cin >> name;

      cout << "Enter the age:";
      cin >> age;
    }
    void show()
    {
      cout << "\n Your name is " << name << "\n Your age is " << age;
    }
};
```

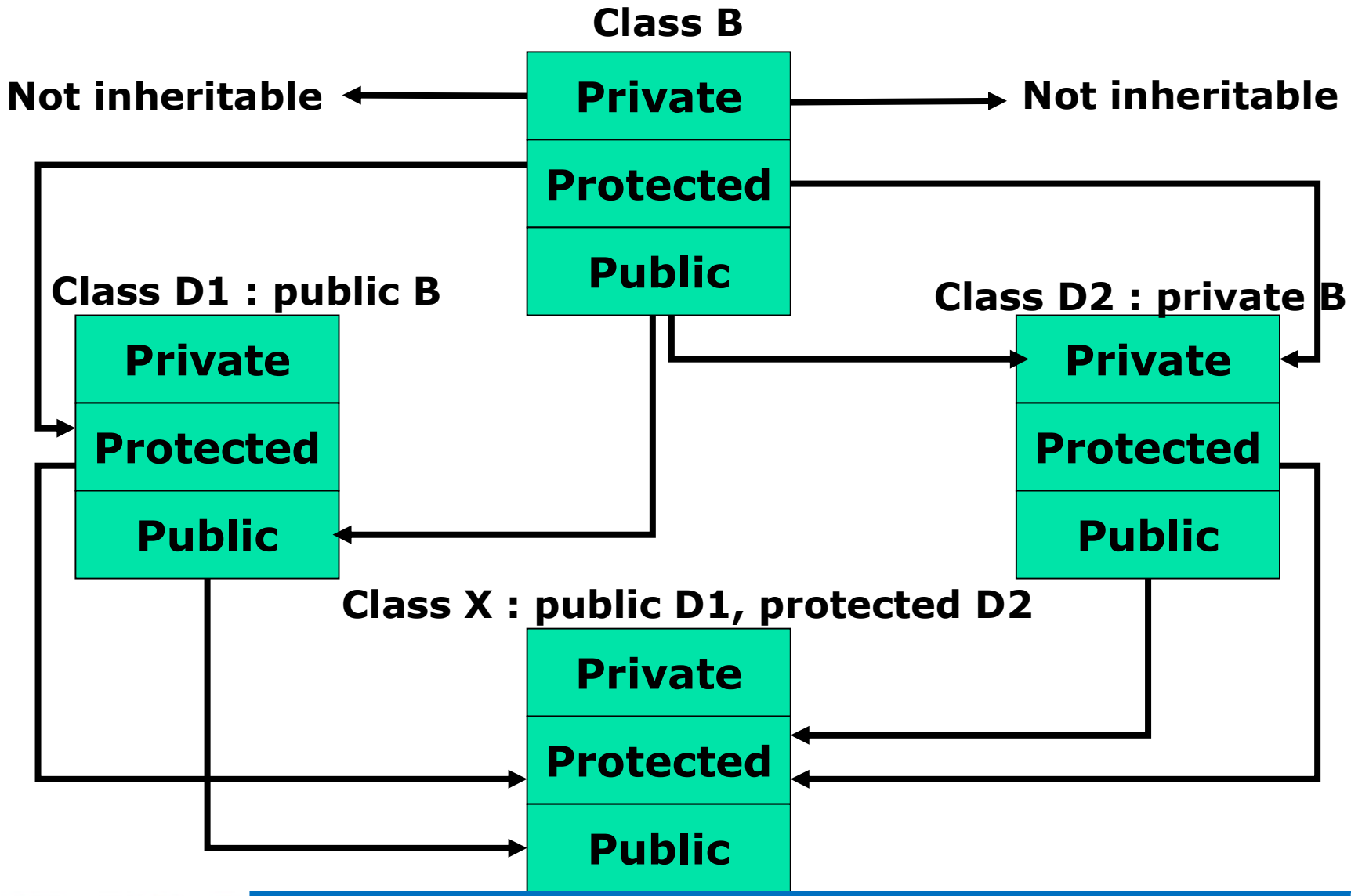
```
class manager : protected worker
// derived class inherit the base class using protected derivation
{
  int now;
  public:
    void get()
    {
      //directly inputing the data
      cout << "Enter your name : ";
      cin << name;
      cout << "Enter your age : ";
      cin >> age;
      cout << "Enter the number of workers under you : ";
      cin >> now;
    }
    void show()
    {
      cout << "\n My name is " << name << "\n My age is " << age;
      cout << "Number of workers under you is " << now;
    }
};
```

```
int main ()
{
   manager m1;
   m1.get();
   cout << "\n \n";
   m1.show();
   return 0;
}
```

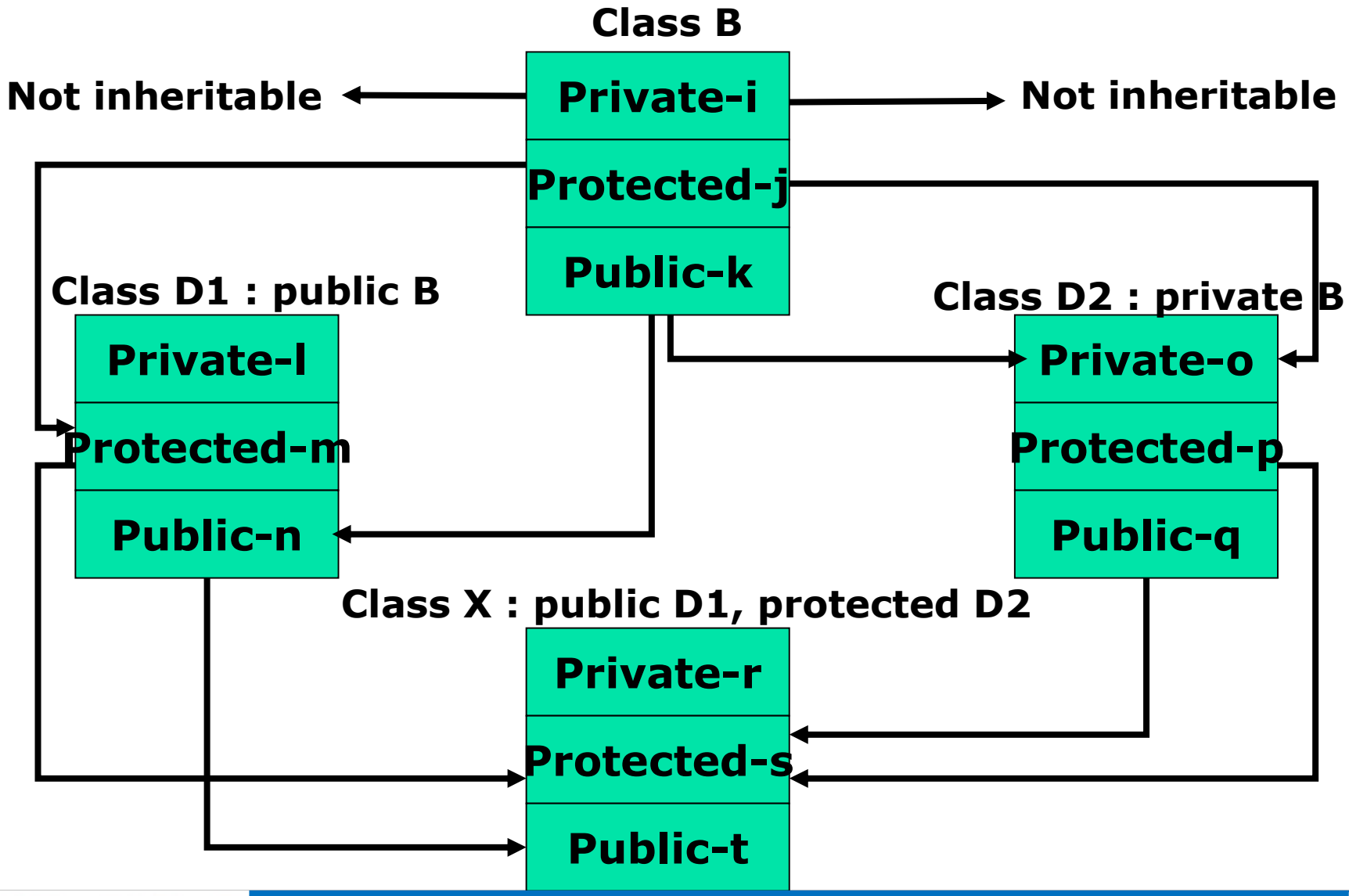
Output ???

This program will give error at the time of compilation as the class is protected and code is trying to access the private components of the class.

Effect of Inheritance on the visibility of Members



Effect of Inheritance on the visibility of Members



Order of key-word

- Private, protected and public members may appear in any order.

```
class beta
{
  protected:
      .....
  public:
      .....
  private:
      .....
  public:
      .....
};
```

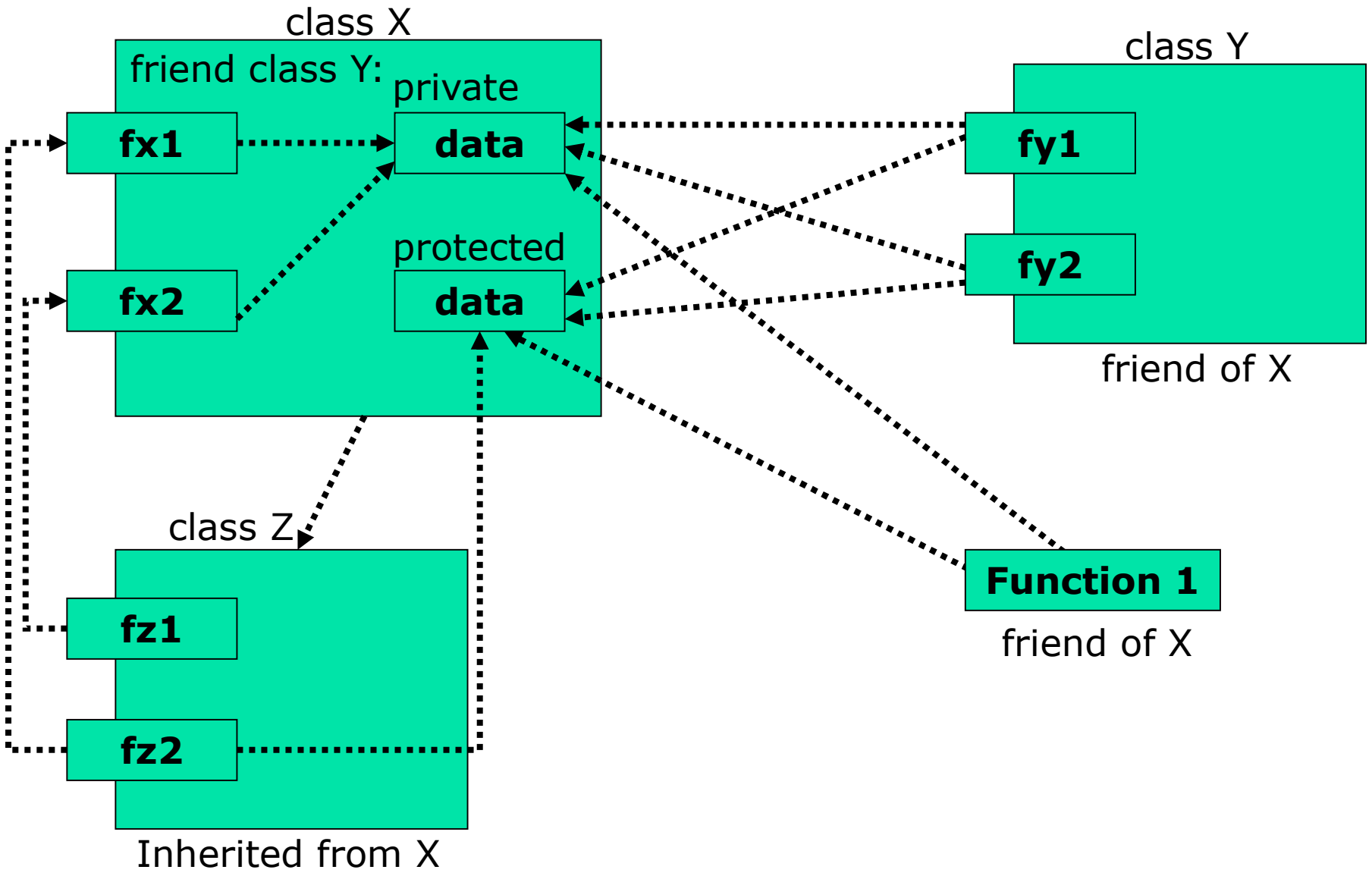
Visibility

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

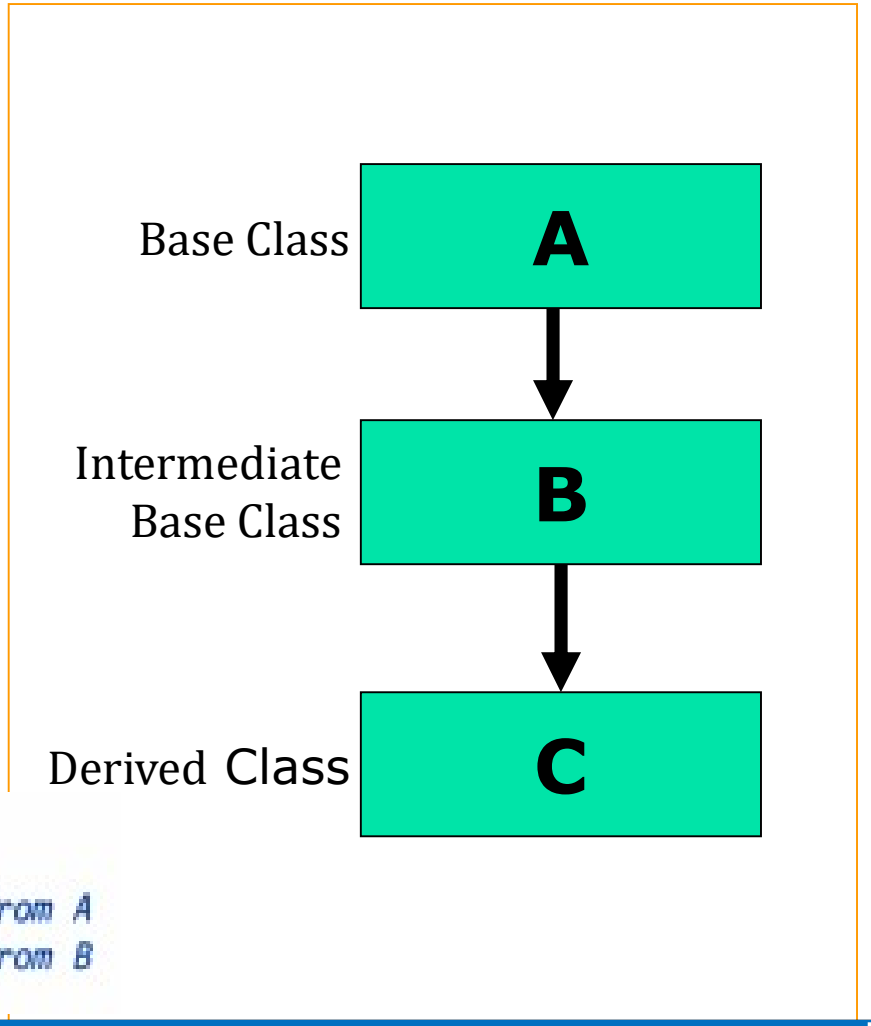
- Functions that can have access to the private and protected members of a class:
  - A function that is a friend of the class.
  - A member function of a class that is a friend of the class.
  - A member function of a derived class.

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 A, B, C is known as inheritance path.




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

Multilevel Inheritance

continue …

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

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

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

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

class father
{
  int age;
  char name [20];

  public:
    void get()
    {
      cout << "Enter your father's Name : ";
      cin >> name;
      cout << "Enter your father's Age : ";
      cin >> age;
    }
    void show()
    {
      cout << "\n Your father's name is " << name;
      cout << "\n Your father's age is " << age;
    }
};
```

```
class mother : public father
{
  int age;
  char name [20];

  public:
    void get()
    {
      cout << "Enter your mother's Name : ";
      cin >> name;
      cout << "Enter your mother's Age : ";
      cin >> age;
    }
    void show()
    {
      cout << "\n Your mother's name is " << name;
      cout << "\n Your mother's age is " << age;
    }
};
```

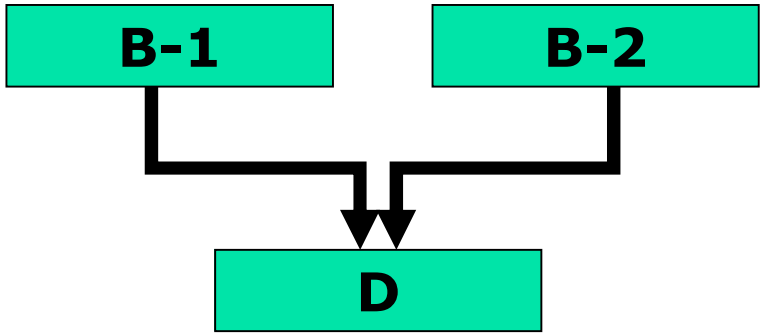
```
class daughter : public mother
{
  int age;
  char name [20];
  public:
    void get()
    {
      father :: get();
      mother :: get();
      cout << "Enter the child's Name : ";
      cin >> name;
      cout << "Enter the child's Age : ";
      cin >> age;
    }
    void show()
    {
      father :: show();
      mother :: show();
      cout << "\n Child's name is " << name;
      cout << "\n Child's age is " << age;
    }
};
int main ()
{
  daughter d1;
  d1.get();
  d1.show();
}
```

Output:

```
Enter your father's Name : abc
Enter your father's Age : 24

Enter your mother's Name : xyz
Enter your mother's Age : 23

Enter the child's Name : abcxyz
Enter the child's Age : 5

Your father's name is abc

Your father's age is 24

Your mother's name is xyz

Your mother's age is 23

Child's name is abcxyz
Child's age is 5
```

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.

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

class father
{
  int age;
  char name [20];

  public:
    void get()
    {
      cout << "Enter your father's name:";
      cin >> name;
      cout << "Enter your father's age:";
      cin >> age;
    }
    void show()
    {
      cout << "\n Your father's name is " << name;
      cout << "\n Your father's age is " << age;
  }
};
```

```
class mother
{
  int age;
  char name [20];

  public:
    void get()
    {
      cout << "Enter your mother's name:";
      cin >> name;
      cout << "Enter your mother's age:";
      cin >> age;
    }
    void show()
    {
      cout << "\n Your mother's name is " << name;
      cout << "\n Your mother's age is " << age;
    }
};
```

```
class daughter : public father, public mother
{
  int s1;
  char name [20];

  public:
    void get()
    {
      father :: get();
      mother :: get();

      cout << "   Enter the child's name:";
      cin >> name;
      cout << "   Enter the child's standard:";
      cin >> s1;
    }
    void show()
    {
      father :: show();
      mother :: show();

      cout << "\n   Child's name is " << name <<endl;
      cout << "\n   Child's age is " << s1 <<endl;
    }
};
```

```
int main ()
{
    daughter d1;
    d1.get();
    d1.show();
}
```

Output:

```
Enter your father's name:xyz
Enter your father's age:27
Enter your mother's name:abc
Enter your mother's age:26
Enter the child's name:abxy
Enter the child's standard:7

Your father's name is xyz
Your father's age is 27
Your mother's name is abc
Your mother's age is 26
Child's name is abxy

Child's age is 7
```

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();}
};

void main( )
{
          P p;
          p.display( );
}
```

In Multiple Inheritance

Ambiguity Resolution in Inheritance

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

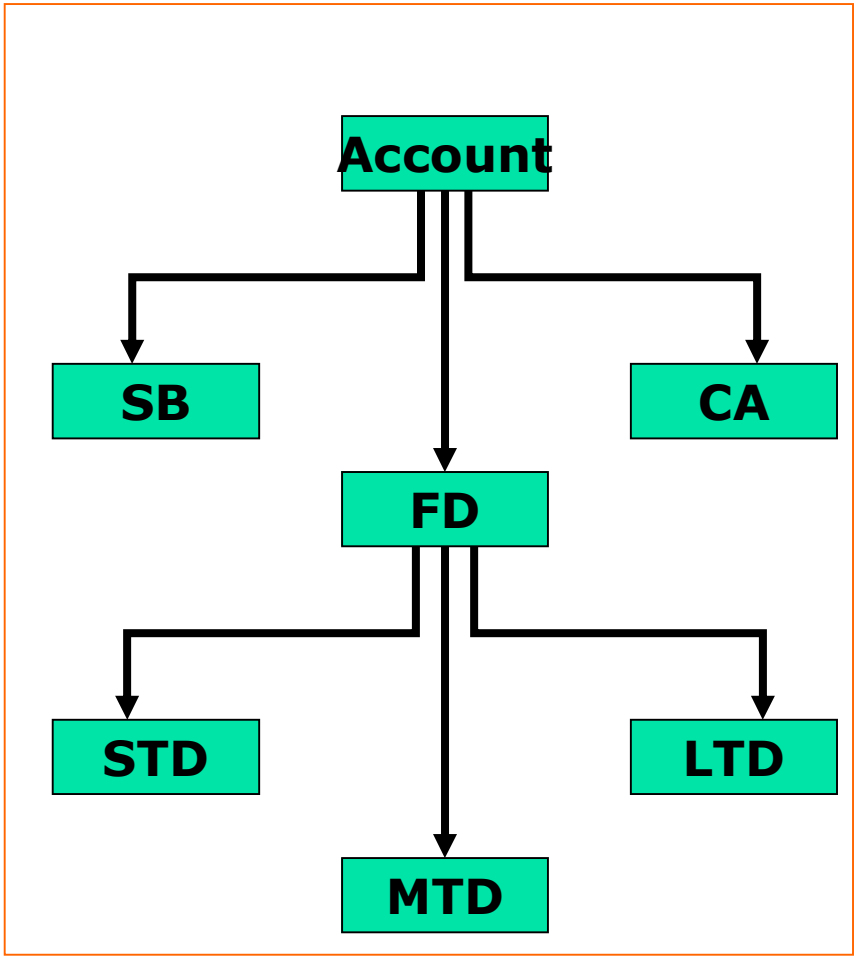
```
void main( )
{
  B b;
  b.display( );      // in B
  b.A::display( );  // in A
  b.B::display( );  // in B
}
```

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

In Single Inheritance

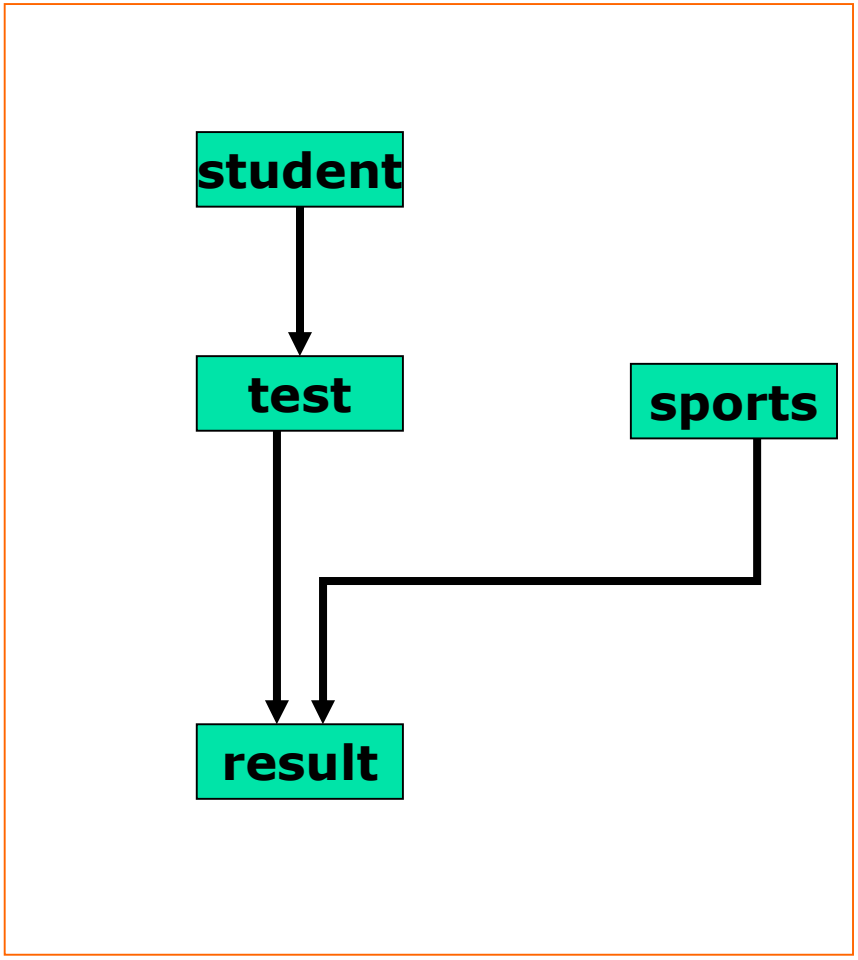
Hierarchical Inheritance

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



Hybrid Inheritance

- Applying Two or more types of inheritance together.



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

class student
//base class derivation
{
  protected:
    int r_no;

  public:
    void getRollno()
    {
      cout << "Enter the roll number of student : ";
      cin >> r_no;
    }

    void putRollno()
    {
      cout << "\nRoll Number -: " << r_no << "\n";
    }
};
```

```
class test : public student
//intermediate base class
{
  protected:
    int part1, part2;

  public:
    void getMarks()
    {
      cout << "Enter the marks of student in SA 1 : ";
      cin >> part1;
      cout << "Enter the marks of student in SA 2 : ";
      cin >> part2;
    }

    void putMarks()
    {
      cout << "Marks Obtained : " << "\n";
      cout << "  Part 1 -: " << part1;
      cout << "\n  Part 2 -: " << part2 << "\n";
    }
};
```

```
class sports
{
  protected:
    int score;

  public:
    void getSportsMarks()
    {
      cout << "Enter the marks in Physical Eduction : ";
      cin >> score;
    }

    void putSportsMarks()
    {
      cout << "Additional Marks : " << score << "\n \n";
    }
};
```

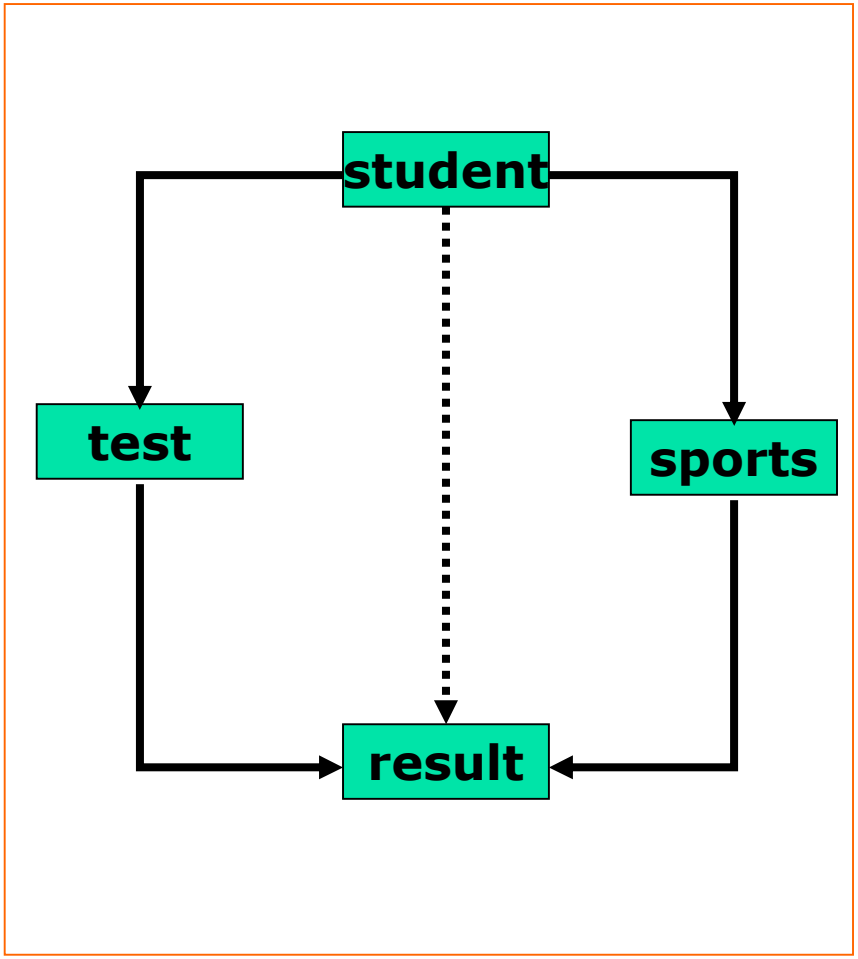
```
class result : public test, public sports
{
  int total;

  public:
    void display ()
    {
      total = part1 + part2 + score;
      putRollno();
      putMarks();
      putSportsMarks();

      cout << "Total Score : " << total ;
    }
};

int main ()
{
  result s1;
  s1.getRollno();
  s1.getMarks();
  s1.getSportsMarks();
  s1.display();

  return 0;
}
```

Output:

```
Enter the roll number of student : 123
Enter the marks of student in SA 1 : 34
Enter the marks of student in SA 2 : 45
Enter the marks in Physical Eduction : 23

Roll Number -: 123
Marks Obtained :
Part 1 -: 34
Part 2 -: 45
Additional Marks : 23

Total Score : 102
```

Virtual Base Classes

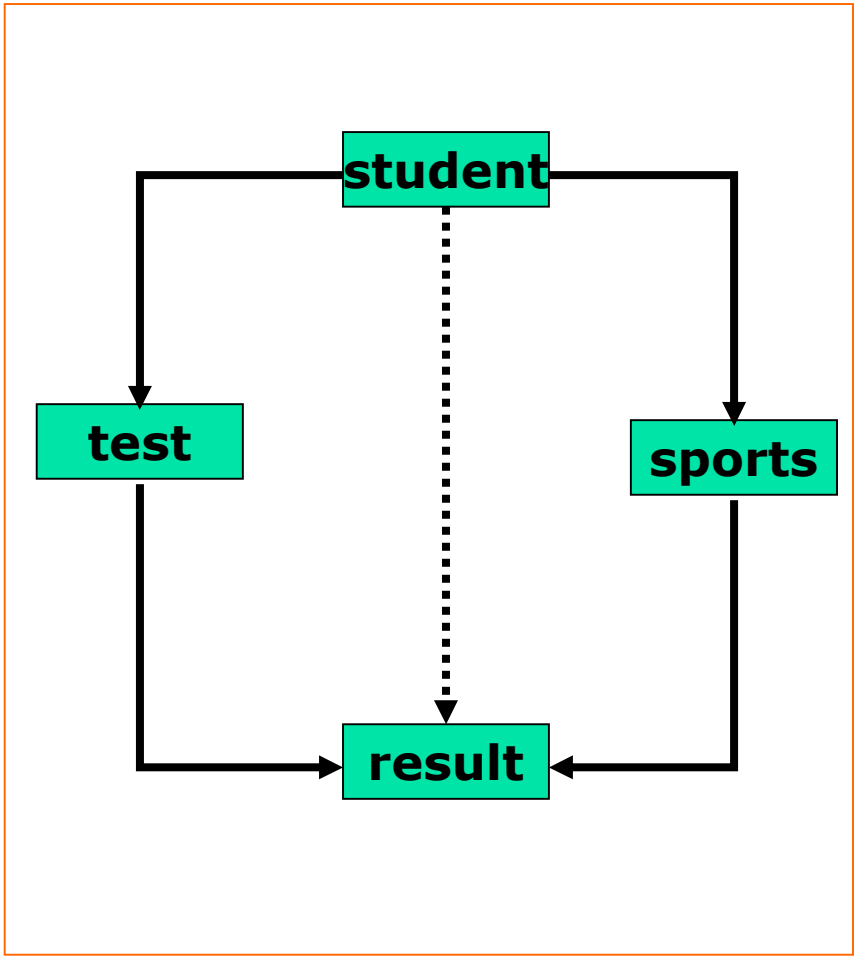
- Here the result class has two direct base classes test and sports which themselves have a common base class student.
- The result inherits the traits of student via two separate paths.



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.

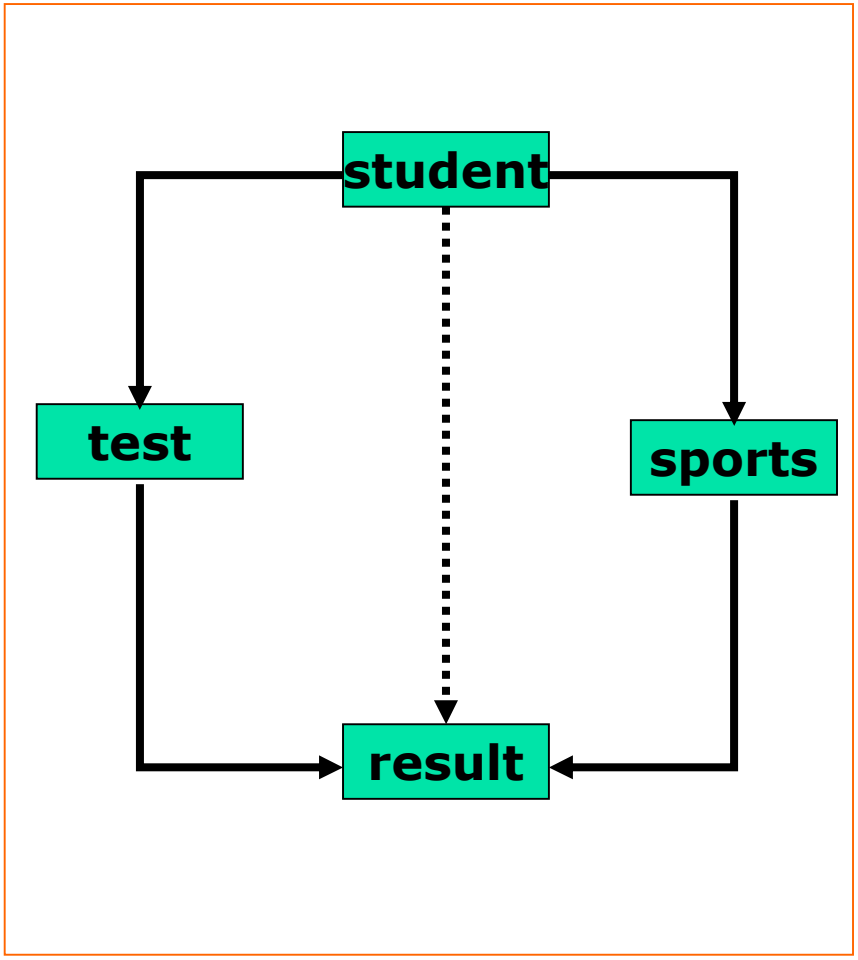


Virtual Base Classes

continue ...

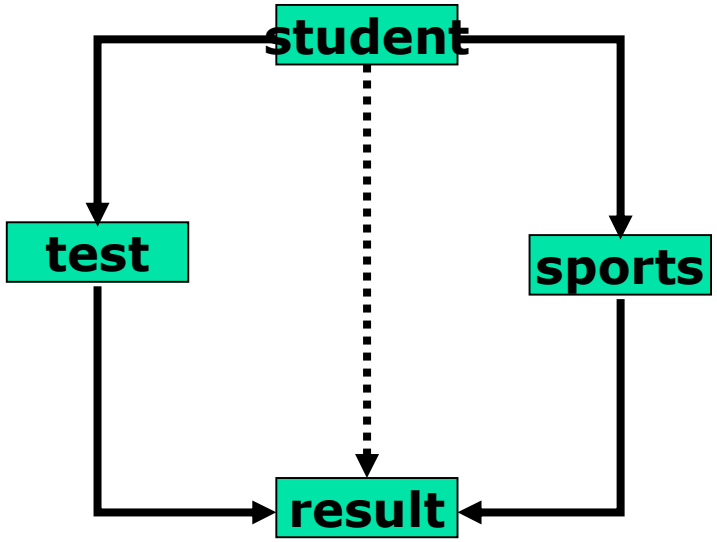
- All the public and protected members of student are inherited into result twice, first via test and again via sports.

- This means result class have duplicate set of members inherited from student.



Virtual Base Classes

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



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

class base
//base class derivation
{
  public:
  int i;
};

class derived1 : virtual public base
// derived1 class inherits base class as virtual.
{
  public:
  int j;
};

class derived2 : virtual public base
{
  public:
  int k;
};

class derived3 : public derived1, public derived2
{
  public:
  int sum;
};
```

note

The keywords **virtual** and **public** may be used in either order.

```
int main ()
{
    derived3 ob;

  ob.i = 10;
  // accessing base class variable

  ob.j = 20;
  // accessing derived1 class variable

  ob.k = 30;
  // accessing derived2 class variable

  ob.sum = ob.i + ob.j + ob.k;
  // accessing its own original member

  cout << "i = " << ob.i ;
  cout << "\nj = " << ob.j ;
  cout << "\nk = " << ob.k ;
  cout << "\nSum of above variables is : " << ob.sum;

  return 0;
}
```

```
i = 10
j = 20
k = 30
Sum of above variables is : 60
```

Output:

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.
- 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

- 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.
- 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

- 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)

{

}

```
D(int a1, int a2, float b1, float b2, int d1):
A(a1, a2),          /* call to constructor A */
B(b1, b2)          /* call to constructor B */
{
     d = d1;       // executes its own body
}
```

```cpp
#include<iostream>
using namespace std;
class Base
{ int x;
  public:
  Base() { cout << "\n   Base default constructor"; }
};

class Derived : public Base
{ int y;
  public:
  Derived() { cout << "\n   Derived default constructor"; }
  Derived(int i) { cout << "\n   Derived parameterized constructor"; }
};

int main()
{
 Base b;
 Derived d1;
 Derived d2(10);
}
```

Output:

```
Base default constructor
Base default constructor
Derived default constructor
Base default constructor
Derived parameterized constructor
```

Order of execution

Method of inheritance	*Order of execution*

Class B: public A
{
};
	A( ) ; base constructor
	B( ) ; derived constructor

class A : public B, public C
{
};
	B( ) ; base(first)
	C( ) ; base(second)
	A( ) ; derived

class A : public B, virtual public C
{
};
	C( ) ; virtual base
	B( ) ; ordinary base
	A( ) ; derived

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

```
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

- 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

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

Presentation Prepared By:

Ms. Khushi Patel

Contact us:

dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
khushipatel.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal

Ms. Krishna Patel

Thank You