AAB University

Faculty of Computer Sciences

Object Oriented Programming

Week 10:

# Inheritance

Asst. Prof. Dr. **Mentor Hamiti**
mentor.hamiti@universitetiaab.com

## *Last Time?!*

- Pointers
- Pointer Operators
- Pass-by-Reference with Pointers
- Built-In Arrays
- Using ***const*** with Pointers
- sizeof Operator
- Pointer Expressions and Pointer Arithmetic
- Relationship Between Pointers and Built-In Arrays

## Today

- Inheritance
- Base & Derived Classes
- Access Control and Type of Inheritance
- Constructor and Inheritance
- Overriding Base Class Functions
- Virtual Base Class

## *Inheritance*

- One of the most important concepts in OOP is that of **inheritance**
    - *Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application*
    - *This also provides an opportunity to reuse the code functionality and fast implementation time*
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class

## *Inheritance*

- Inheritance is a form of software reuse in which you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities
  - You can designate that the **new class** should inherit the members of an existing class
  - This existing class is called the **base class**, and the new class is referred to as the **derived class**
  - A derived class represents a more specialized group of objects
- C++ offers public, protected and private inheritance
  - *However, base-class objects are not objects of their derived classes*

## Inheritance

- **Base classes** tend to be **more general** and **derived classes** tend to be **more specific**

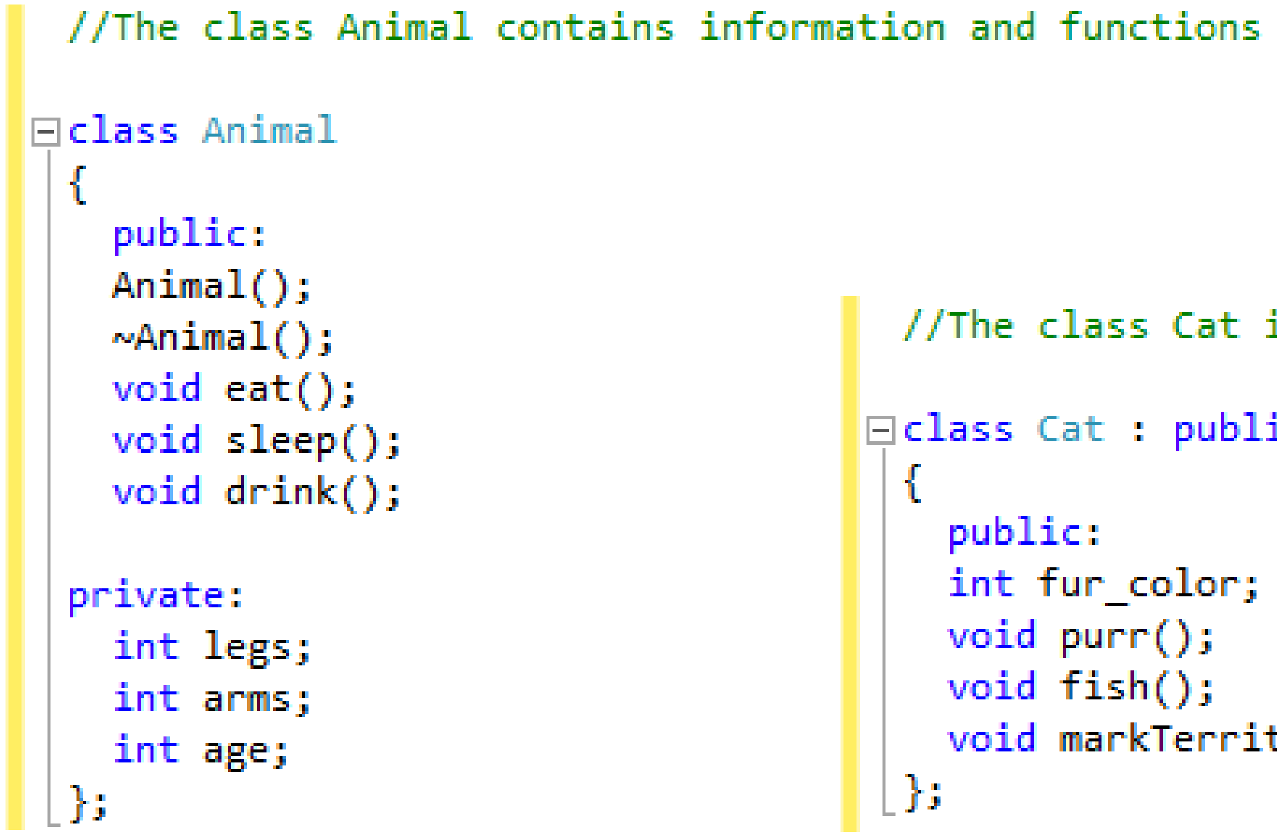
| Base class | Derived classes |
|---|---|
| Student | GraduateStudent, UndergraduateStudent |
| Shape | Circle, Triangle, Rectangle, Sphere, Cube |
| Loan | CarLoan, HomeImprovementLoan, MortgageLoan |
| Employee | Faculty, Staff |
| Account | CheckingAccount, SavingsAccount |

- *Because every derived-class object is an object of its base class, and one base class can have many derived classes, the set of objects represented by a base class typically is larger than the set of objects represented by any of its derived classes*

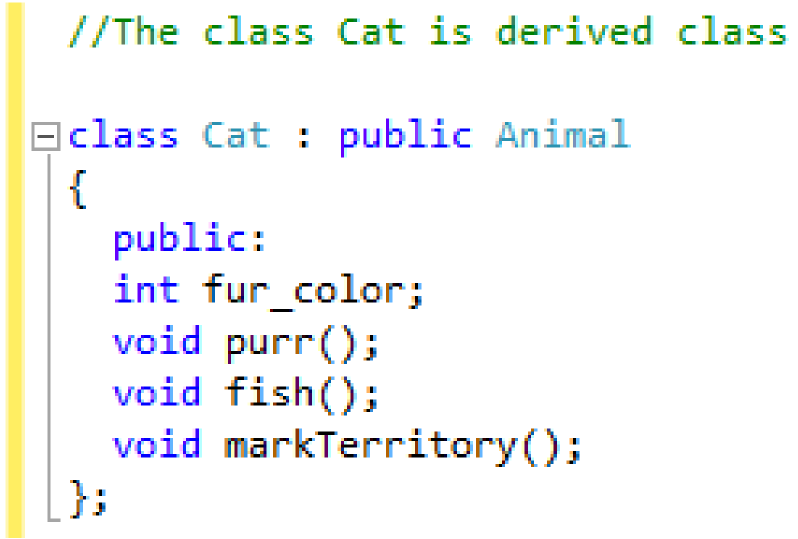
## *Inheritance*

- **Base classes** tend to be **more general** and **derived classes** tend to be **more specific**

- Example:



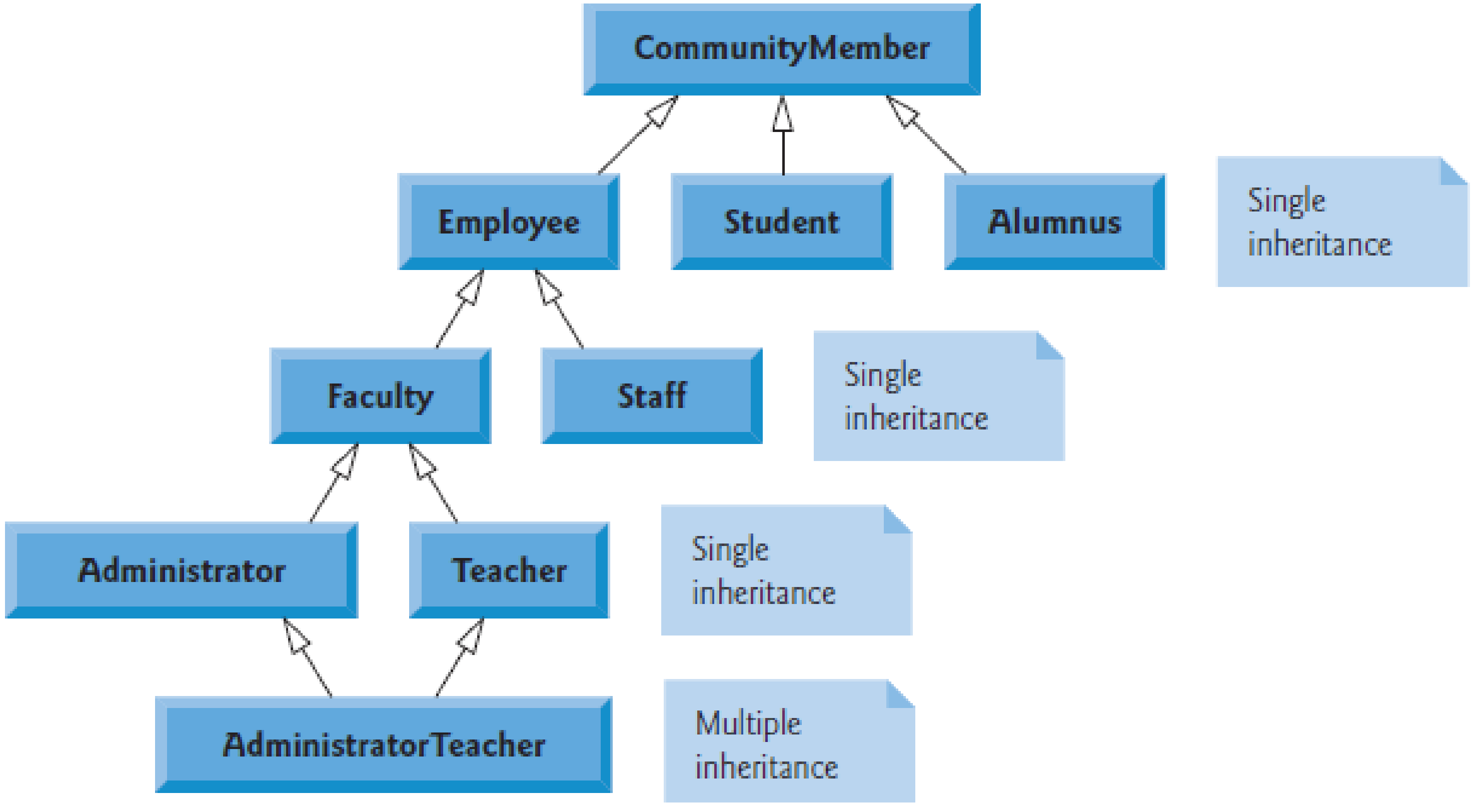
```
//The class Animal contains information and functions

class Animal
{
  public:
  Animal();
  ~Animal();
  void eat();
  void sleep();
  void drink();

private:
  int legs;
  int arms;
  int age;
};
```



```
//The class Cat is derived class

class Cat : public Animal
{
  public:
  int fur_color;
  void purr();
  void fish();
  void markTerritory();
};
```

## Inheritance

- Example: Inheritance Hierarchy for University Community Members

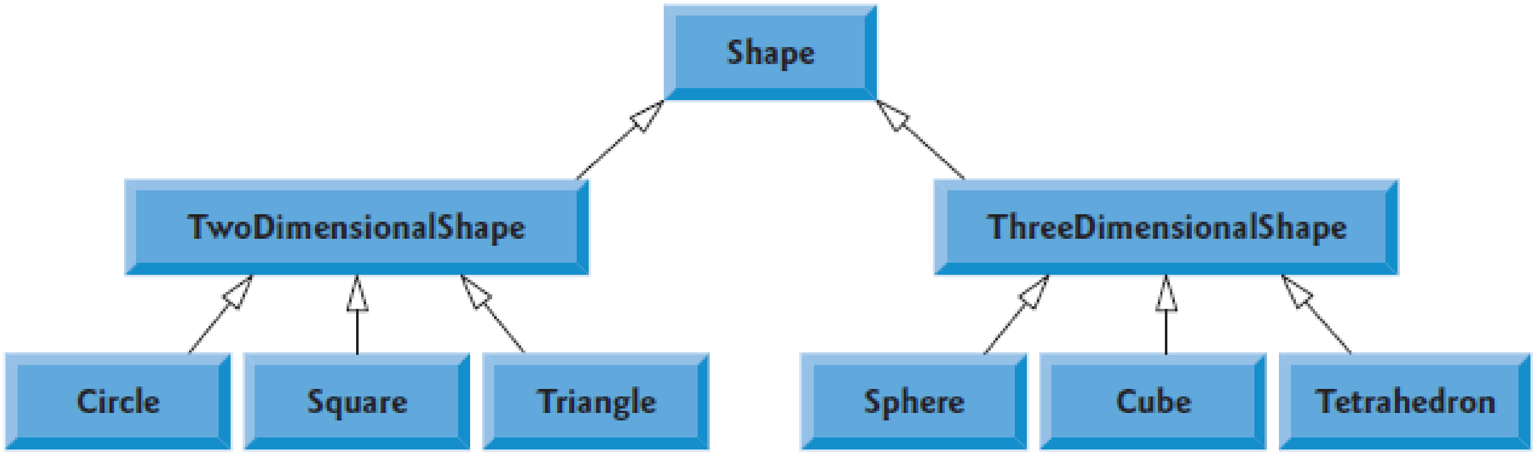


## *Inheritance*

- Each **arrow** in the hierarchy represents an relationship
  - As we follow the arrows in this class hierarchy, we can state "an Employee is a CommunityMember" and "a Teacher is a Faculty member"
  - CommunityMember is the direct base class of Employee, Student and Alumnus
  - CommunityMember is an indirect base class of all the other classes in the diagram
- With **single inheritance**, a class is derived from one base class
- With **multiple inheritance**, a derived class inherits simultaneously from two or more base classes

# Inheritance

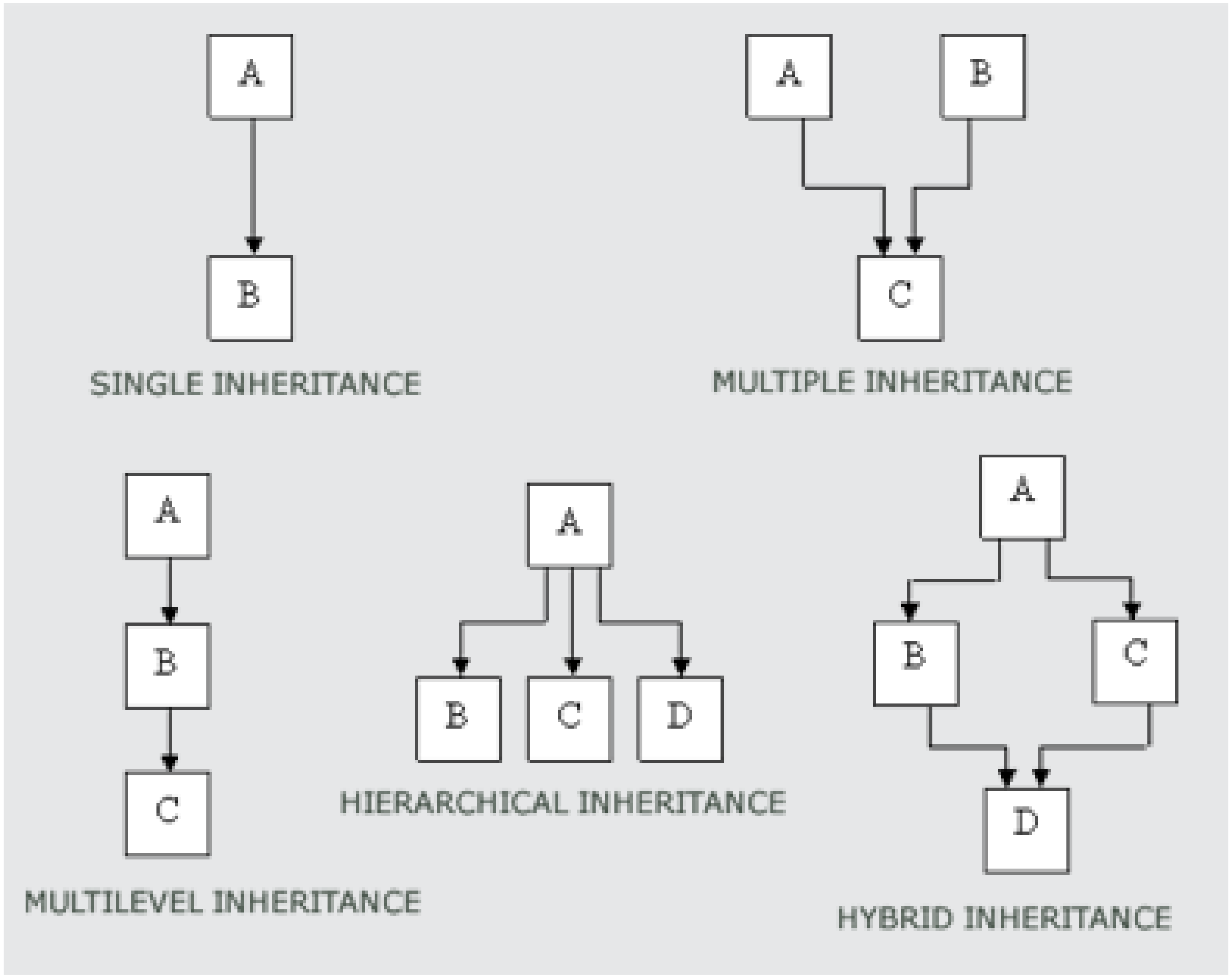
- Example: Shape Class Hierarchy



- As showed, we can follow the arrows from the bottom of the diagram to the topmost base class in this class hierarchy to identify several relationships
  - *Begins with base class Shape*
  - *Classes TwoDimensionalShape and ThreeDimensionalShape derive from base class*
  - *The third level of this hierarchy contains some more specific types of TwoDimensionalShape and ThreeDimensionalShapes*

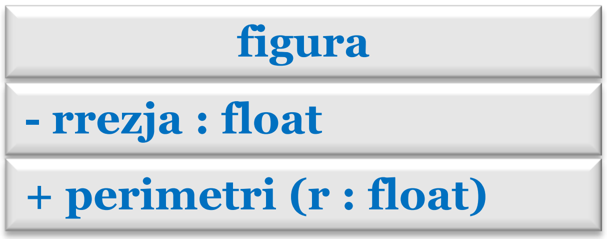
# *Inheritance*

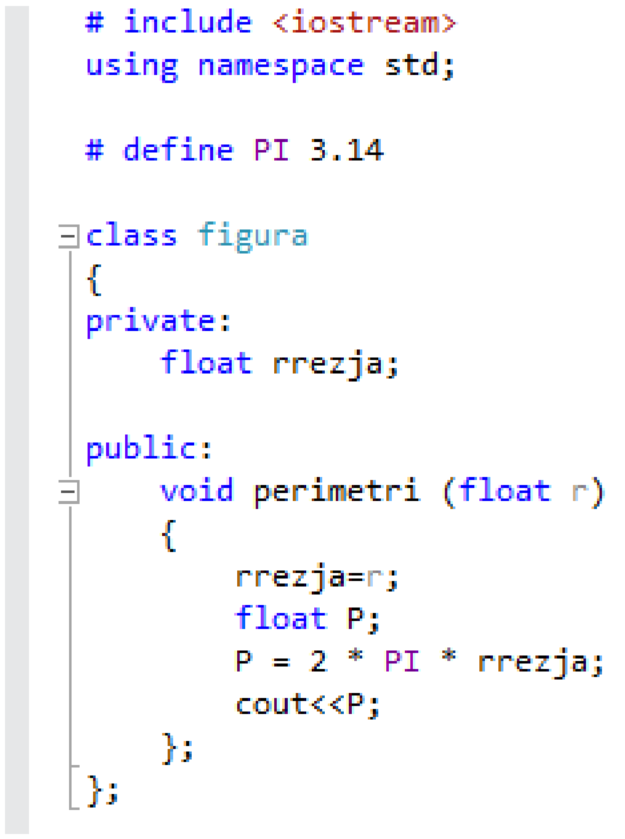
- Forms of Inheritance



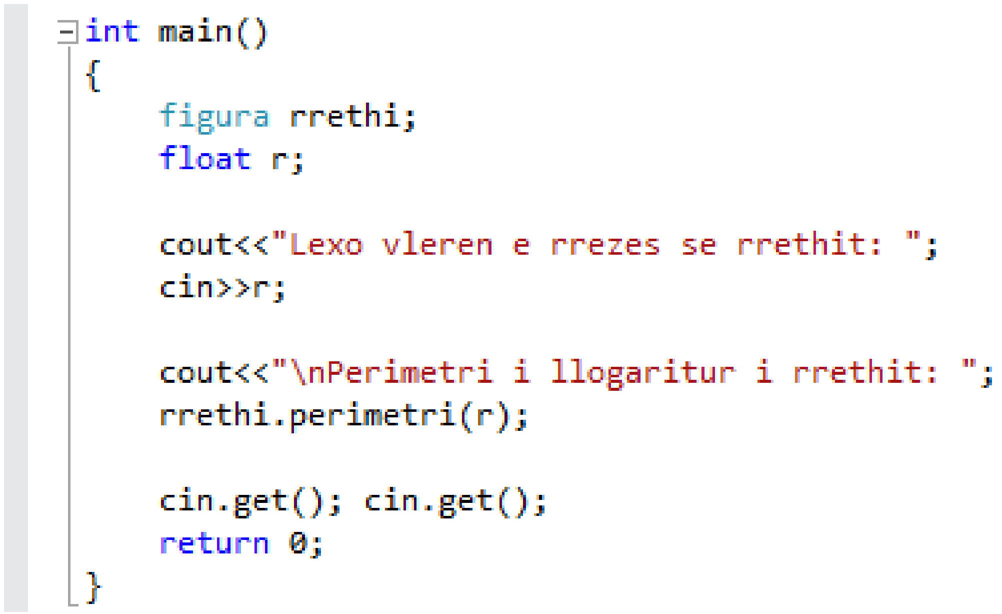
## Base Classes

- Example 1:

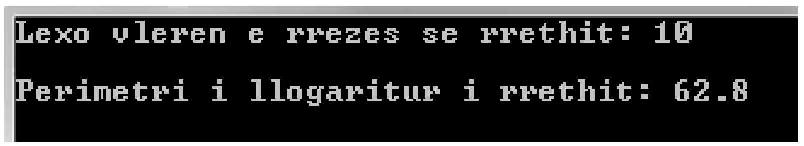




```cpp
# include <iostream>
using namespace std;

# define PI 3.14

class figura
{
private:
    float rrezja;

public:
    void perimetri (float r)
    {
        rrezja=r;
        float P;
        P = 2 * PI * rrezja;
        cout<<P;
    };
};
```



```cpp
int main()
{
    figura rrethi;
    float r;

    cout<<"Lexo vleren e rrezes se rrethit: ";
    cin>>r;

    cout<<"\nPerimetri i llogaritur i rrethit: ";
    rrethi.perimetri(r);

    cin.get(); cin.get();
    return 0;
}
```



```
Lexo vleren e rrezes se rrethit: 10

Perimetri i llogaritur i rrethit: 62.8
```

## *Base & Derived Classes*

- **A class** can be derived from more than one classes, which means it can inherit data and functions from multiple base classes

- To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form:

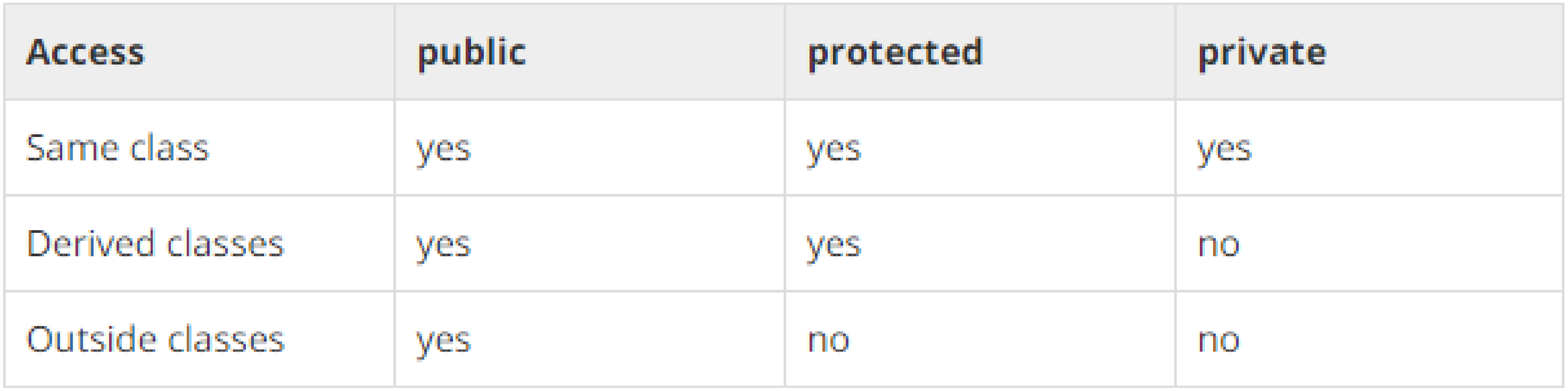
```
class derived-class:  access-specifier base-class
```

- Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default

## *Access Control and Inheritance*

- A **derived class** can access <u>all the non-private members of its base class</u>
    - *Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class*

- We can summarize the different access types according to who can access them in the following way:



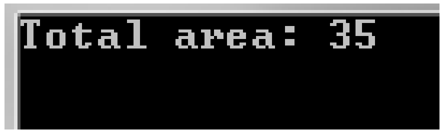
| Access | public | protected | private |
|---|---|---|---|
| Same class | yes | yes | yes |
| Derived classes | yes | yes | no |
| Outside classes | yes | no | no |

## Access Control and Inheritance

- Example 2: A base class Shape and its derived class Rectangle

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

// Base class
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setHeight(int h)
      {
         height = h;
      }
   protected:
      int width;
      int height;
};
```

```cpp
// Derived class
class Rectangle: public Shape
{
   public:
      int getArea()
      {
         return (width * height);
      }
};

int main(void)
{
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   cin.get(); return 0;
}
```



```
Total area: 35
```

## *Type of Inheritance*

- When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance
  - *The type of inheritance is specified by the access-specifier*

- We hardly use protected or private inheritance, but public inheritance is commonly used.

- While using different type of inheritance, following rules are applied:

## *Type of Inheritance*

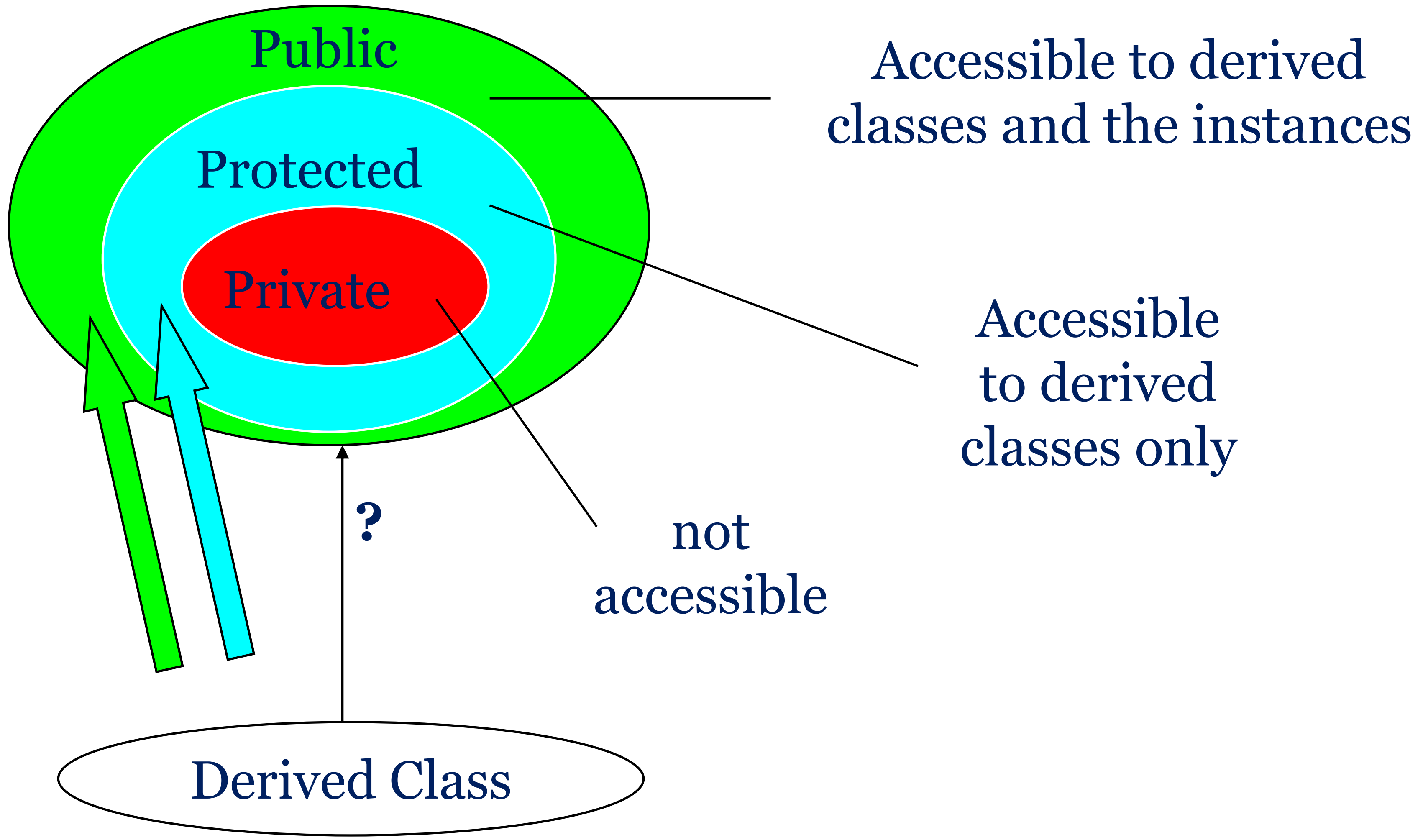
- **Public Inheritance**: When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class

- **Protected Inheritance**: When deriving from a protected base class, public and protected members of the base class become protected members of the derived class

- **Private Inheritance**: When deriving from a private base class, public and protected members of the base class become private members of the derived class

## Type of Inheritance

- Class Members


Public
Protected
Private
Accessible to derived classes and the instances
Accessible to derived classes only
not accessible
?
Derived Class

## *Multiple Inheritances*

- A **C++** class can **inherit** members from more than one class and here is the extended syntax:

```
class derived-class:  access baseA, access baseB....
```

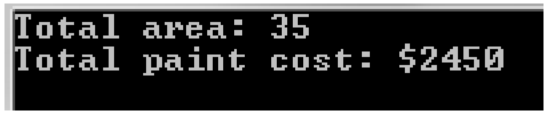
- Where access is one of **public**, **protected**, or **private** and would be given for every base class and they will be separated by comma as shown above

# Multiple Inheritances

- Example 3:

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

// Base class Shape
class Shape
{
    public:
        void setWidth(int w)
        {
            width = w;
        }
        void setHeight(int h)
        {
            height = h;
        }
    protected:
        int width;
        int height;
};

// Base class PaintCost
class PaintCost
{
    public:
        int getCost(int area)
        {
            return area * 70;
        }
};
```

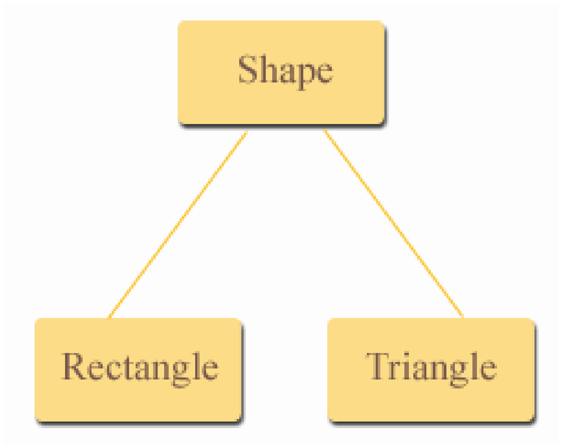
```cpp
// Derived class
class Rectangle: public Shape, public PaintCost
{
    public:
        int getArea()
        {
            return (width * height);
        }
};

int main(void)
{
    Rectangle Rect;
    int area;

    Rect.setWidth(5);
    Rect.setHeight(7);

    area = Rect.getArea();

    // Print the area of the object.
    cout << "Total area: " << Rect.getArea() << endl;

    // Print the total cost of painting
    cout << "Total paint cost: $" << Rect.getCost(area) << endl;

    cin.get(); return 0;
}
```

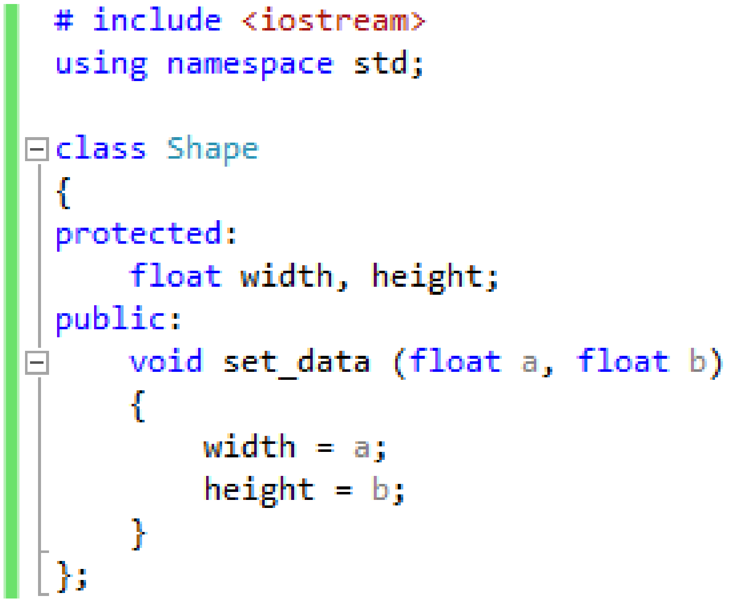


```
Total area: 35
Total paint cost: $2450
```

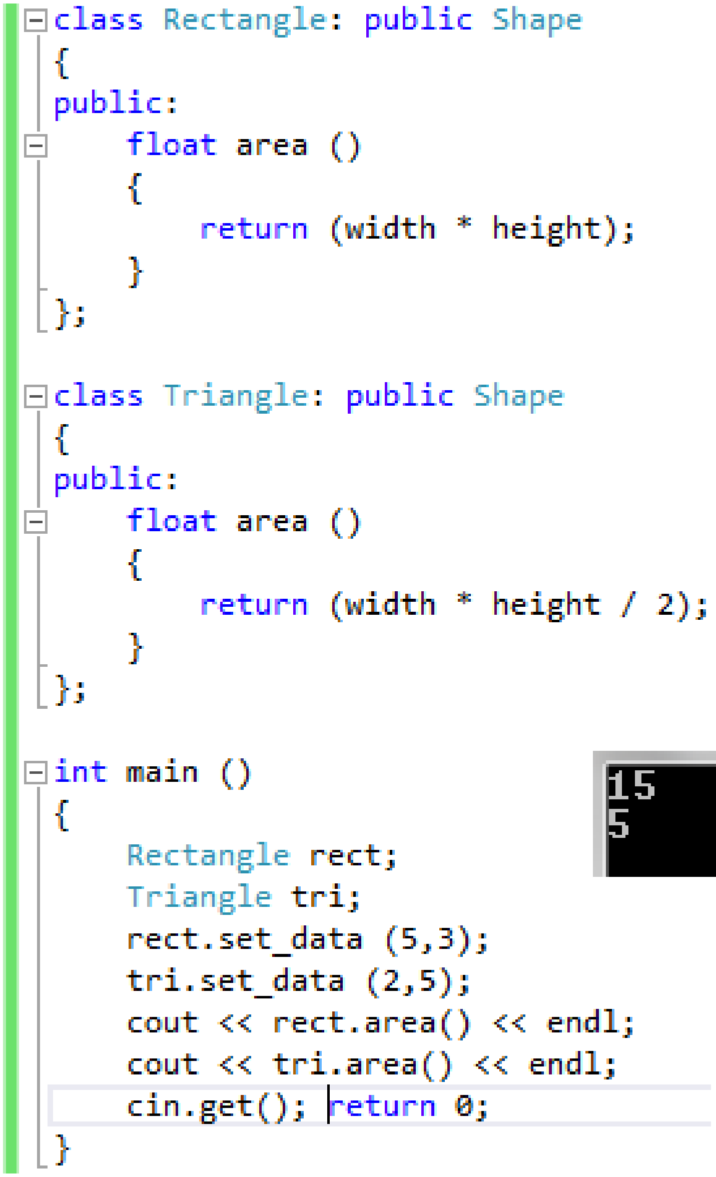
# Inheritances

- Example 4:

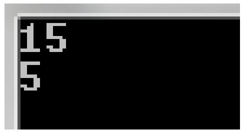




```
# include <iostream>
using namespace std;

class Shape
{
protected:
    float width, height;
public:
    void set_data (float a, float b)
    {
        width = a;
        height = b;
    }
};
```



```
class Rectangle: public Shape
{
public:
    float area ()
    {
        return (width * height);
    }
};

class Triangle: public Shape
{
public:
    float area ()
    {
        return (width * height / 2);
    }
};

int main ()
{
    Rectangle rect;
    Triangle tri;
    rect.set_data (5,3);
    tri.set_data (2,5);
    cout << rect.area() << endl;
    cout << tri.area() << endl;
    cin.get(); return 0;
}
```



```
15
5
```

## Inheritances

- Example 5:

```
class Shape
{
  protected:
        int width, height;
  public:
        void setDims (int a, int b){
        width=a; height=b;}
};
```

```
class Triangle: public Shape
{
  public:
        int area ( ) {
        return (width * height/2); }
};
```

```
class Rectangle: public Shape
{
   public:
        int area ( ) {
        return (width * height); }
};
```

```
class Square: public Rectangle
{
   public:
        void setDims (int a){
        width=a; height=a;}
};
```

# Inheritances

- Example 5:

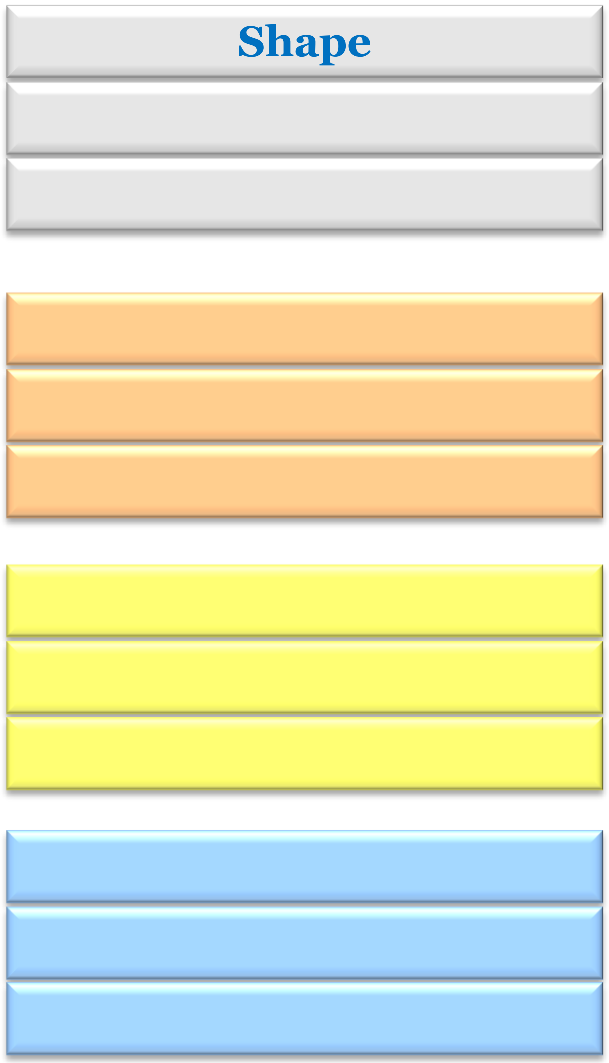
```
class Shape
{
  protected:
       int width, height;
  public:
       void setDims (int a, int b){
       width=a; height=b;}
};
```

```
class Rectangle: public Shape
{
   public:
       int area ( ) {
       return (width * height); }
};
```

```
class Triangle: public Shape
{
  public:
       int area ( ) {
       return (width * height/2); }
};
```

```
class Square: public Rectangle
{
   public:
       void setDims (int a){
       width=a; height=a;}
};
```

**UML class diagram**

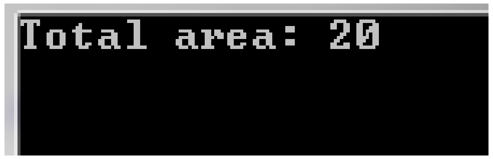


# Inheritances

- Example 5:

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

class Shape
{
    protected:
          int width, height;
    public:
           void setDims (int a, int b){
           width=a; height=b;}
};

class Rectangle: public Shape  {
     public:
           int area ( ) {
           return (width * height); }
};

class Triangle: public Shape  {
    public:
          int area ( ) {
          return (width * height/2); }
};

class Square: public Rectangle  {
     public:
          void setDims (int a){
          width=a; height=a;}
};
```

```cpp
int main(void)
{
   Rectangle Rect;

   Rect.setDims (4, 5);

  // Print the area of the object
  cout << "Total area: " << Rect.area() << endl;

   cin.get();
   return 0;
}
```



```
Total area: 20
```

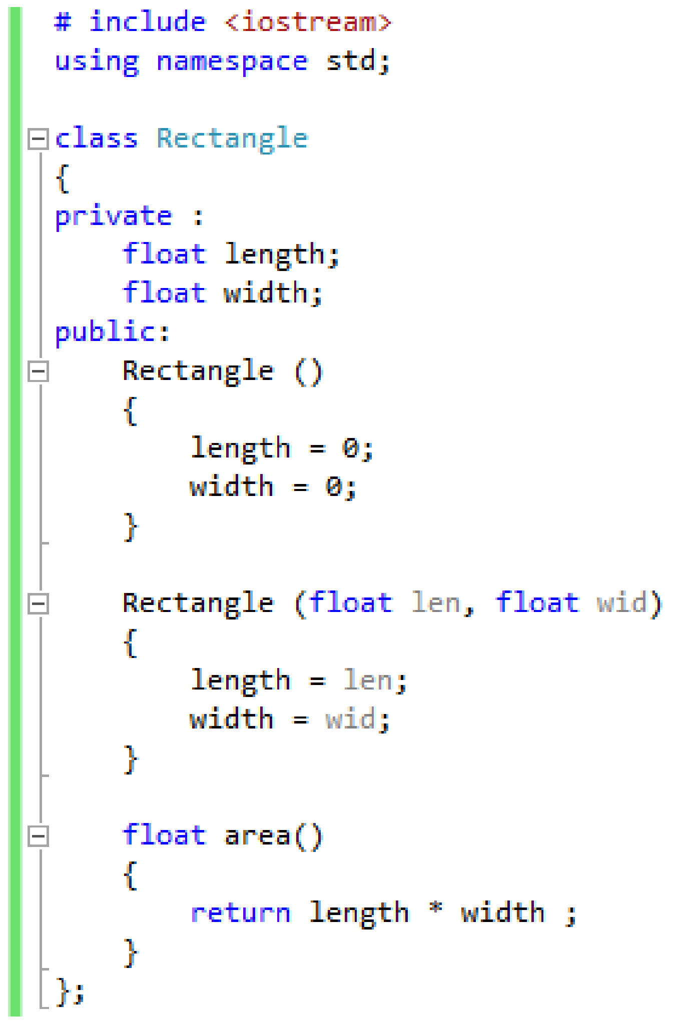
## *Constructor and Inheritance*

- The compiler automatically call a base class constructor before executing the derived class constructor

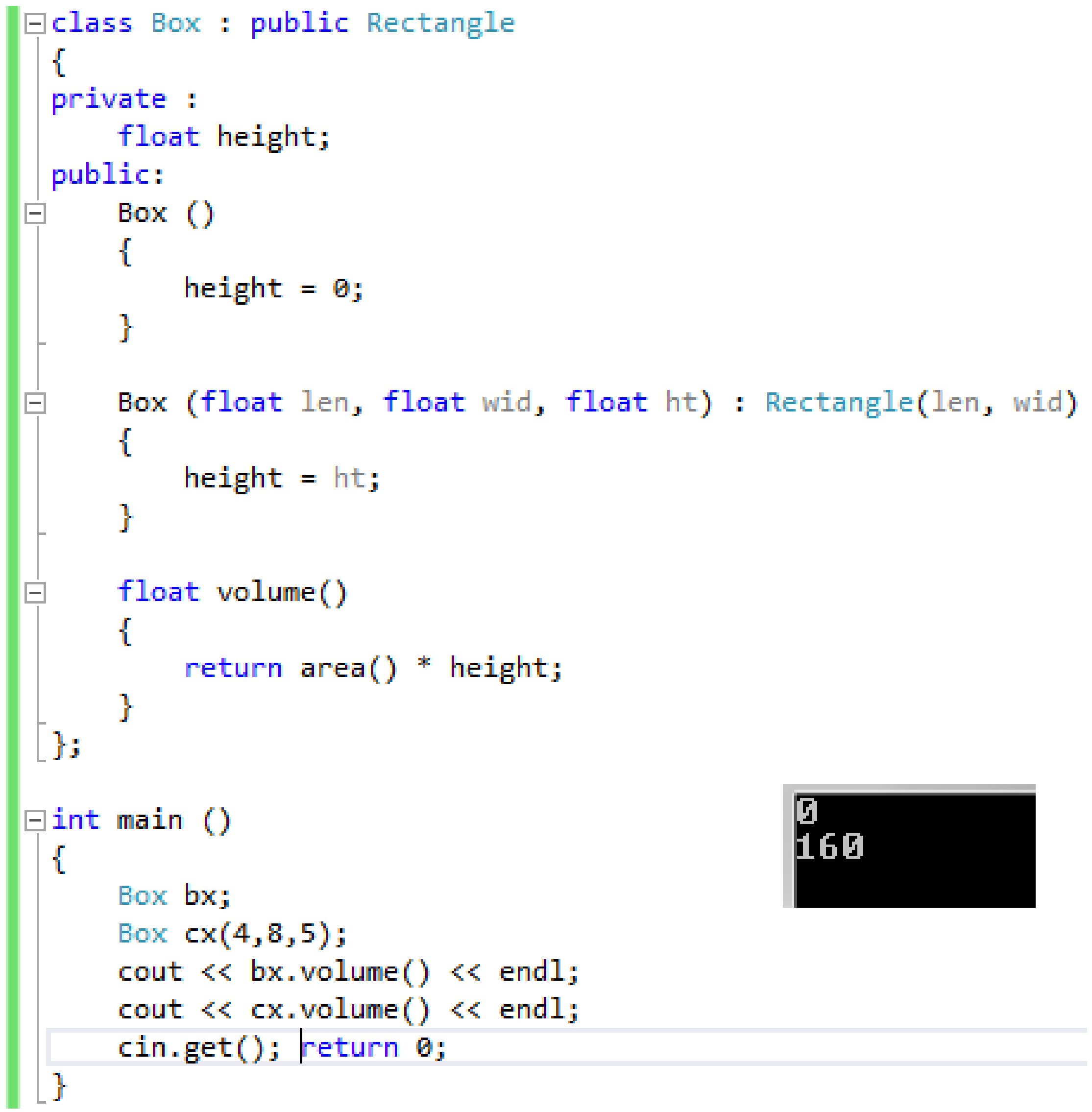
- In these cases, you must explicitly specify which base class constructor should be called by the compiler

## Constructor and Inheritance

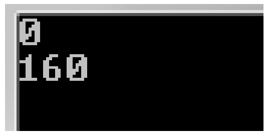
- Example 6:



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

class Rectangle
{
private :
    float length;
    float width;
public:
    Rectangle ()
    {
        length = 0;
        width = 0;
    }

    Rectangle (float len, float wid)
    {
        length = len;
        width = wid;
    }

    float area()
    {
        return length * width ;
    }
};
```



```cpp
class Box : public Rectangle
{
private :
    float height;
public:
    Box ()
    {
        height = 0;
    }

    Box (float len, float wid, float ht) : Rectangle(len, wid)
    {
        height = ht;
    }

    float volume()
    {
        return area() * height;
    }
};

int main ()
{
    Box bx;
    Box cx(4,8,5);
    cout << bx.volume() << endl;
    cout << cx.volume() << endl;
    cin.get(); return 0;
}
```



```
0
160
```

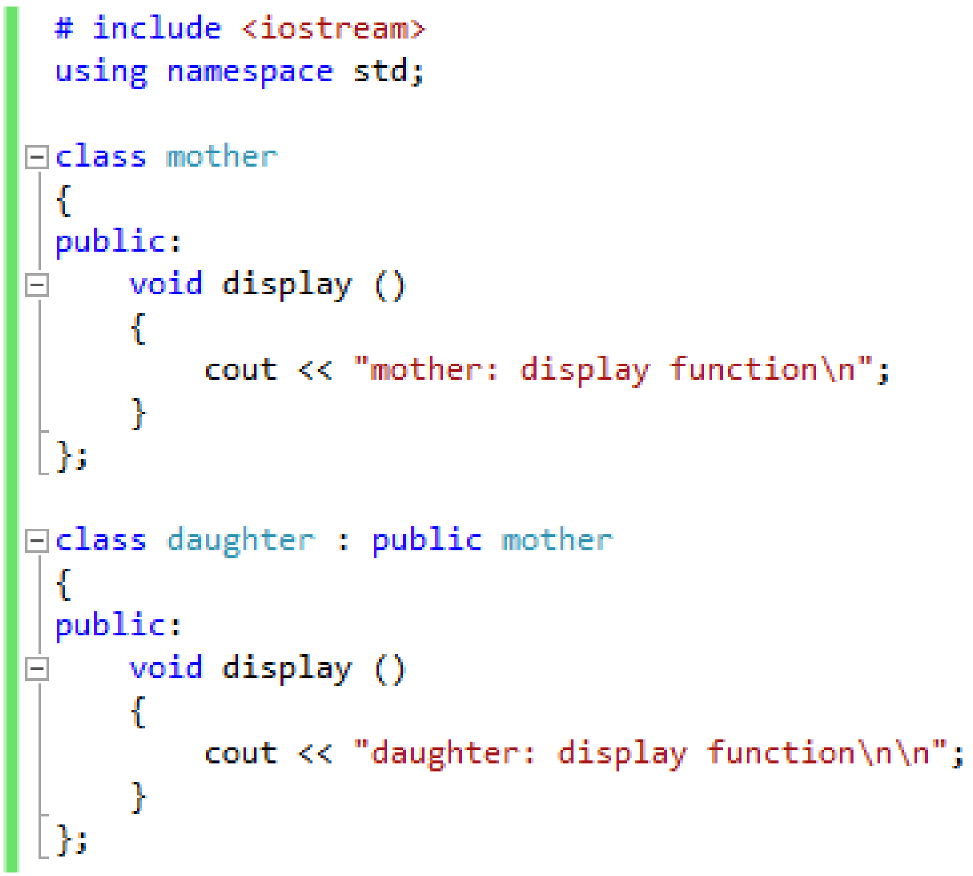
## *Overriding Base Class Functions*

- A derived class can override a member function of its base class by defining a derived class member function with the same name and parameter list

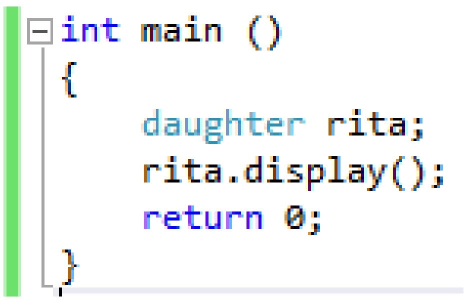
- It is often useful for a derived class to define its own version of a member function inherited from its base class
  - *This may be done to specialize the member function to the needs of the derived class. When this happens, the base class member function is said to be* ***overridden*** *by the derived class*

# Overriding Base Class Functions

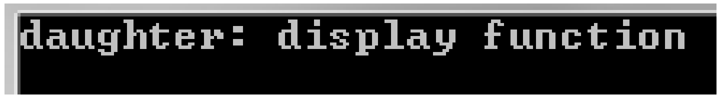
- Example 7:



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

class mother
{
public:
	void display ()
	{
		cout << "mother: display function\n";
	}
};

class daughter : public mother
{
public:
	void display ()
	{
		cout << "daughter: display function\n\n";
	}
};
```



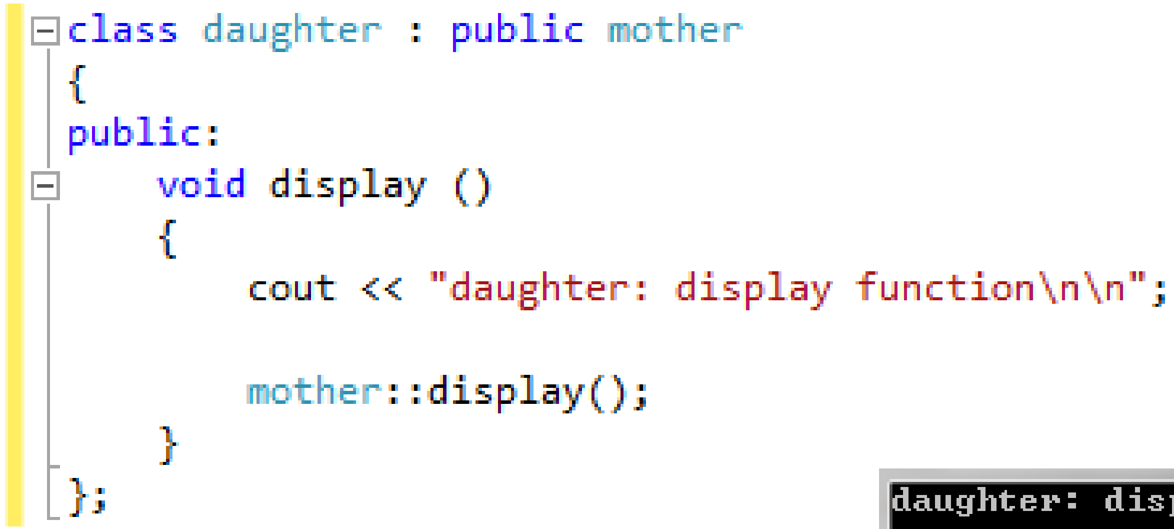
```cpp
int main ()
{
	daughter rita;
	rita.display();
	return 0;
}
```



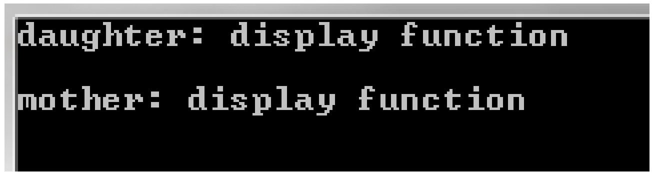
```
daughter: display function
```

## *Gaining Access to an Overridden Function*

- It is occasionally useful to be able to call the overridden version
  - *This is done by using the scope resolution operator to specify the class of the overridden member function being accessed*



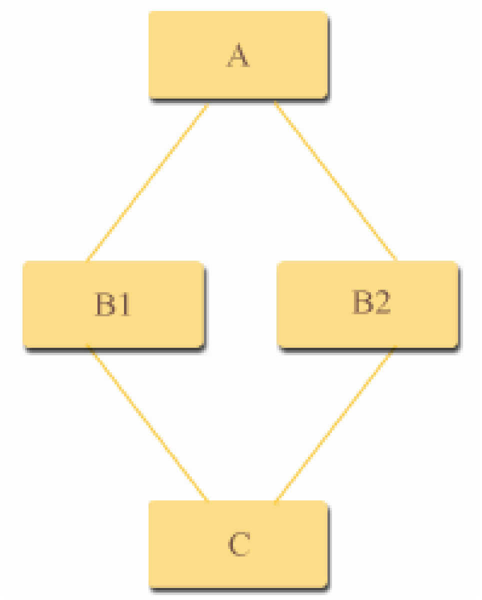
```
class daughter : public mother
{
public:
    void display ()
    {
        cout << "daughter: display function\n\n";

        mother::display();
    }
};
```



```
daughter: display function

mother: display function
```

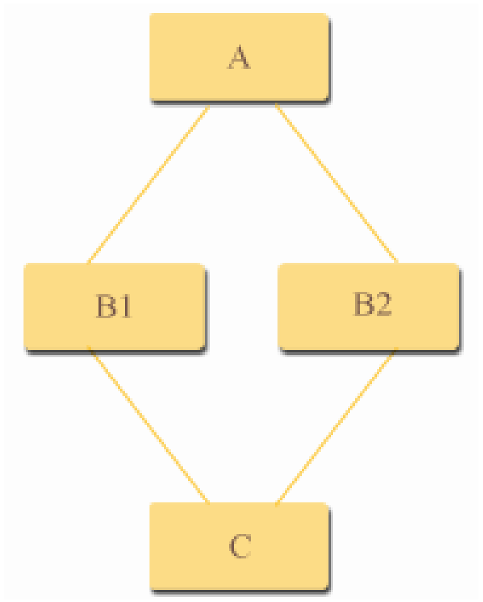
## *Virtual Base Class*

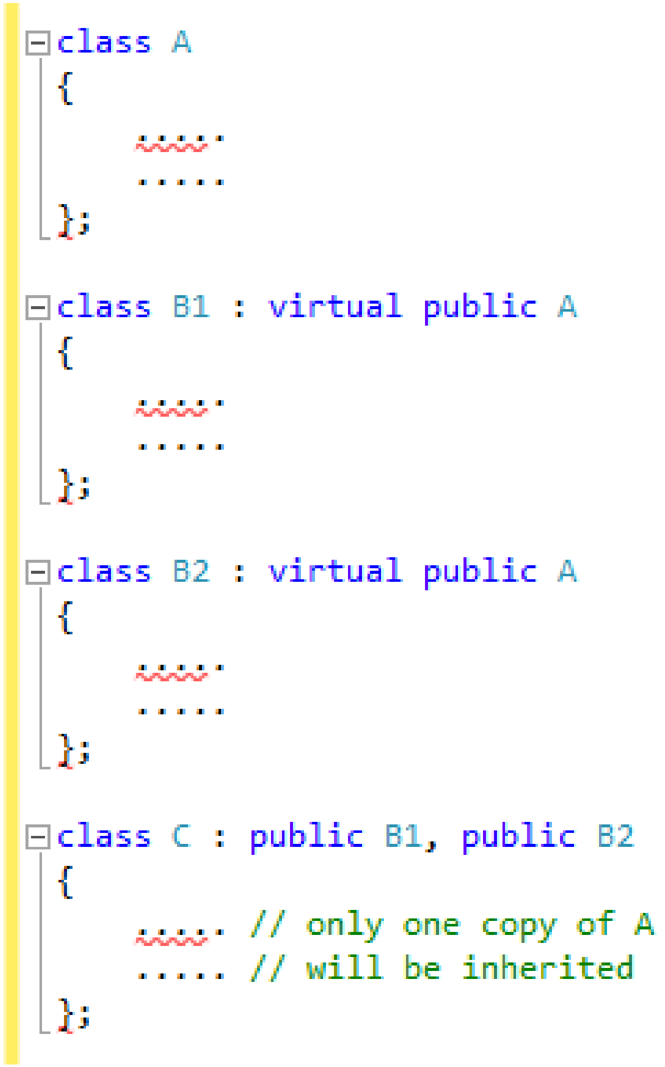
- Multipath inheritance may lead to **duplication** of inherited members from a grandparent base class
  - *This may be avoided by making the common base class a virtual base class. When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited*



## Virtual Base Class

- Example:





```
class A
{
    ......
    ......
};

class B1 : virtual public A
{
    ......
    ......
};

class B2 : virtual public A
{
    ......
    ......
};

class C : public B1, public B2
{
    ...... // only one copy of A
    ...... // will be inherited
};
```

# *Object Oriented Programming*

- Questions?!

mentor.hamiti@universitetiaab.com