# Inheritance

Comp Sci 1575 Data Structures

# Inheritance



"The great thing about Object Oriented code is that it can make small, simple problems look like large, complex ones."

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Inheritance



- When creating a class, rather than writing completely new data members and member functions, make a new class inherit members of an existing class
- Existing class whose properties are inherited by new class is called the **Parent**, **Base**, or **Super class**
- New class which inherits properties of base class is called **Child**, **Derived**, or **Sub class**
- The derived class inherits the members of the base class, on top of which it can add its own members.
- **Inheritance** is the capability of one class to acquire properties and characteristics from another class

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Purpose of inheritance



1. **Code re-usability:** When a new class inherits an existing class, all its methods and fields become available in the new class
2. Makes it easier to create and maintain large applications
3. Allows **abstract classes as interfaces** (more coming up soon)

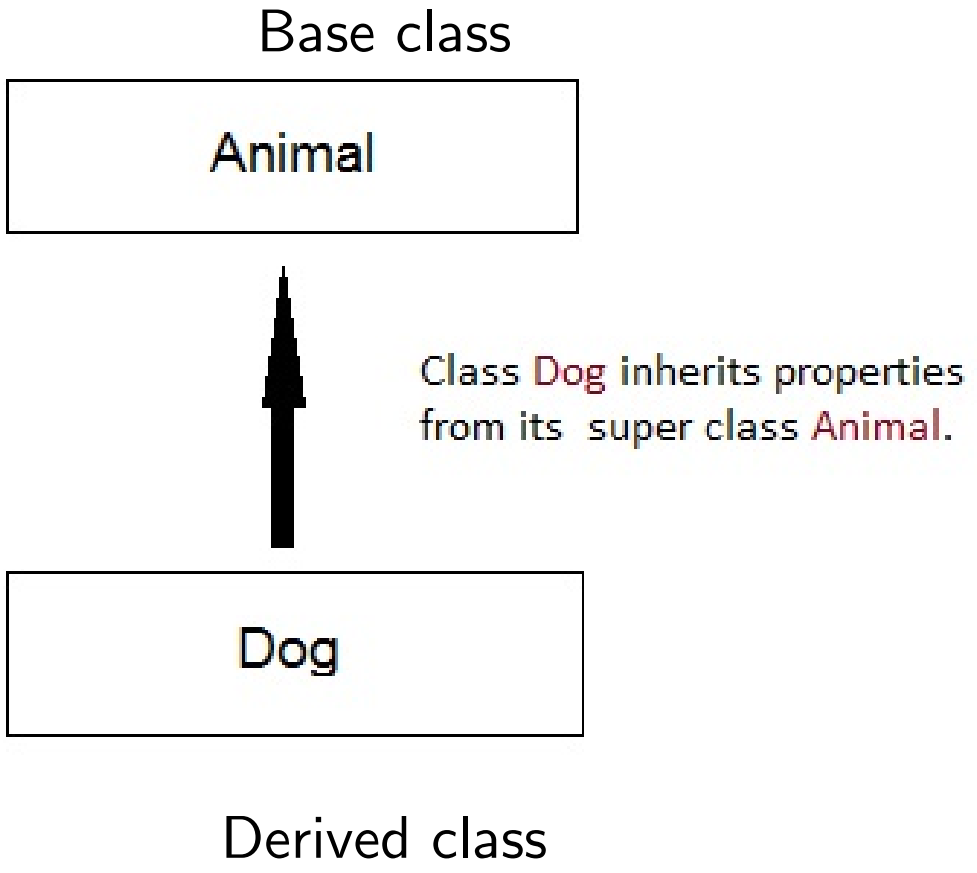
# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Base and derived classes





# Class derivation list specifies the base class



```
class Animal {
  public:
    int brain = 1;
};

// Class derivation list:
class Dog: public Animal {
  public:
    int spots = 34;
};

int main() {
  Dog d;
  cout << d.brain;   // 1
  cout << d.spots;   // 34
}
```

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Access specifiers in base class



Access levels in the original base class:

1. **Public**: Base class's public members are accessible to all
2. **Protected**: Base class's protected members are accessible only to the derived class(s)
3. **Private**: Base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected member functions of the base class.

# Access control modes



Modifiers of the original access specifiers:

1. **Public**: 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. **This is most common.**
2. **Protected**: public and protected members of the base class are demoted to protected members of the derived class
3. **Private**: public and protected members of the base class are demoted to private members of the derived class

# Access control modes modify access specifiers



Original access specifier in base:

| **Who has access?** | **public** | **protected** | **private** |
|---|---|---|---|
| members of the same class | yes | yes | yes |
| members of derived class | yes | yes | no |
| non members | yes | no | no |

---

Base class has members that are public, protected, and private (left index)

| | **Derived Class** | **Derived Class** | **Derived Class** |
|---|---|---|---|
| **Base class** | **Public Mode** | **Protected Mode** | **Private Mode** |
| **Public** | Public | Protected | Private (accessible) |
| **Protected** | Protected | Protected | Private (accessible) |
| **Private** | Not accessible | Not accessible | Not accessible |

Derived class (top indices) inherits these levels (intersections)

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# public mode



```
class Animal {
  string thoughts;
  public:
    int brain = 1;
};

class Dog : public Animal {
  public:
    int spots = 34;
    void printSpots() {cout << brain;} // ??
};

int main() {
  Dog d;
  cout << d.brain; // ??
  d.printSpots(); // ??
  cout << d.spots;  // 34
  cout << d.thoughts; // ??
}
```

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# protected mode



```
class Animal {
  public:
    int brain = 1;
};

// Class derivation list:
class Dog : protected Animal {
  public:
    int spots = 34;
    void printSpots() {cout << brain;} // ??
};

int main() {
 Dog d;
 cout << d.brain; // ??
 d.printSpots(); ??
 cout << d.spots;  // 34
}
```

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# private mode



```
class Animal {
  public:
    int brain = 1;
};

// Class derivation list:
class Dog : private Animal {
  public:
    int spots = 34;
    void printSpots() {cout << brain;} // ??
};

int main() {
  Dog d;
  cout << d.brain; // ??
  d.printSpots(); ??
  cout << d.spots;  // 34
}
```

# Kahoot

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Outline

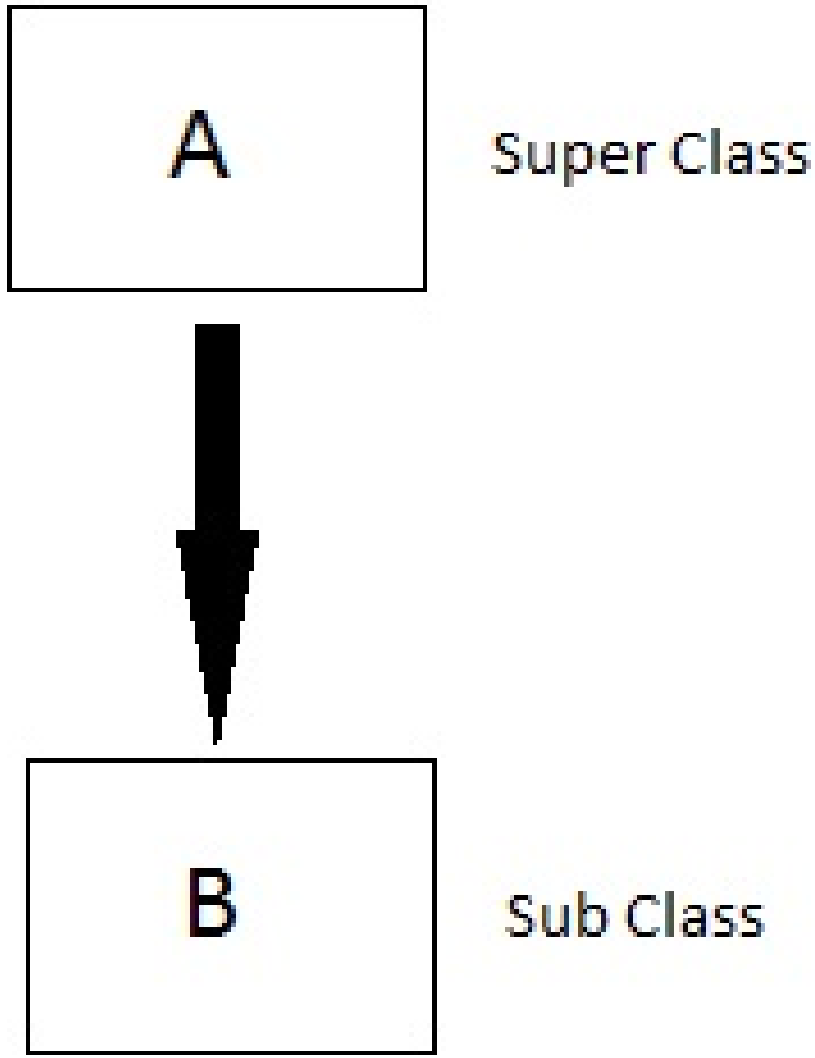



Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Types of inheritance: Single



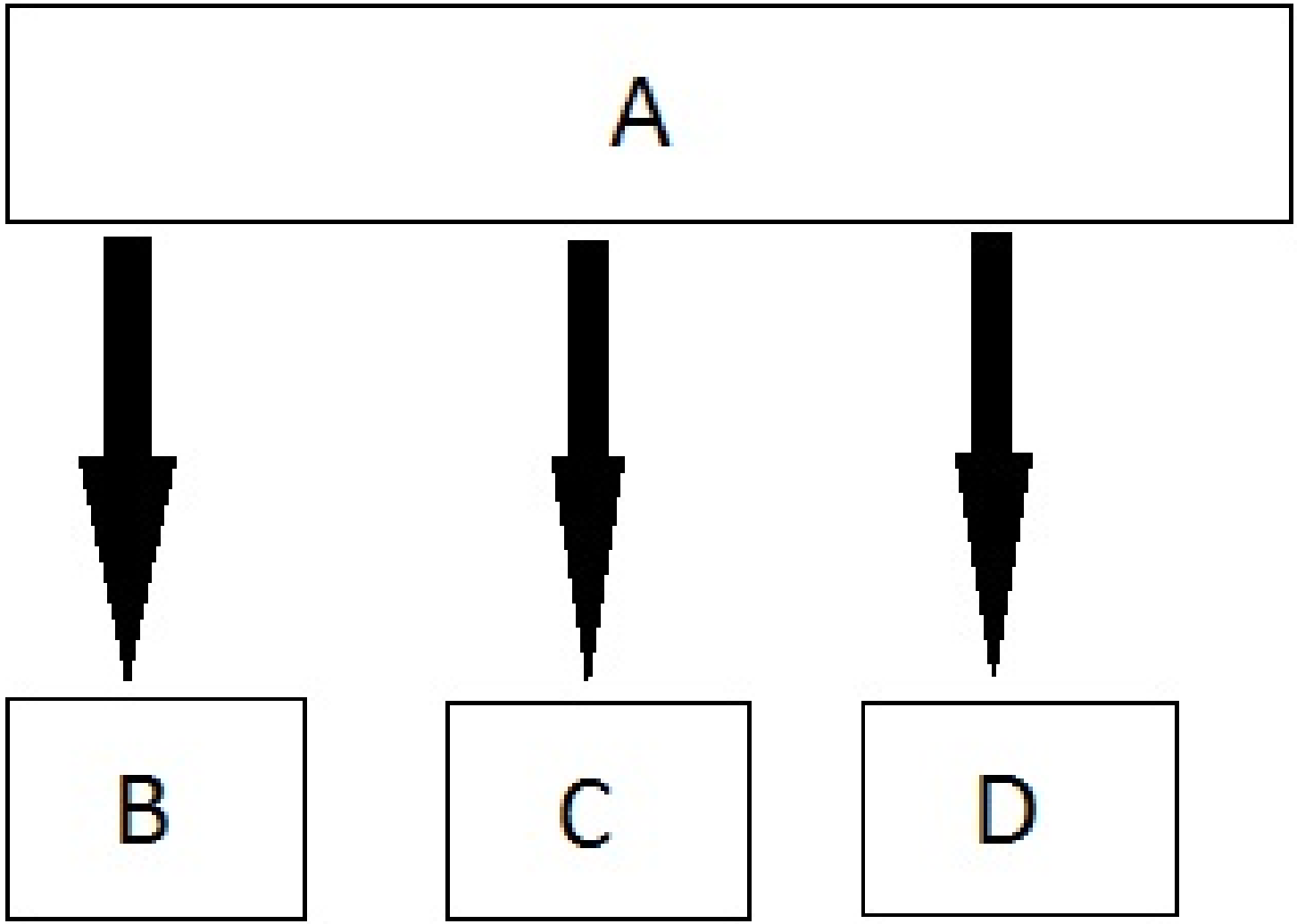
One derived class inherits from only one base class. It is the most simplest form of Inheritance.



# Types of inheritance: Hierarchical

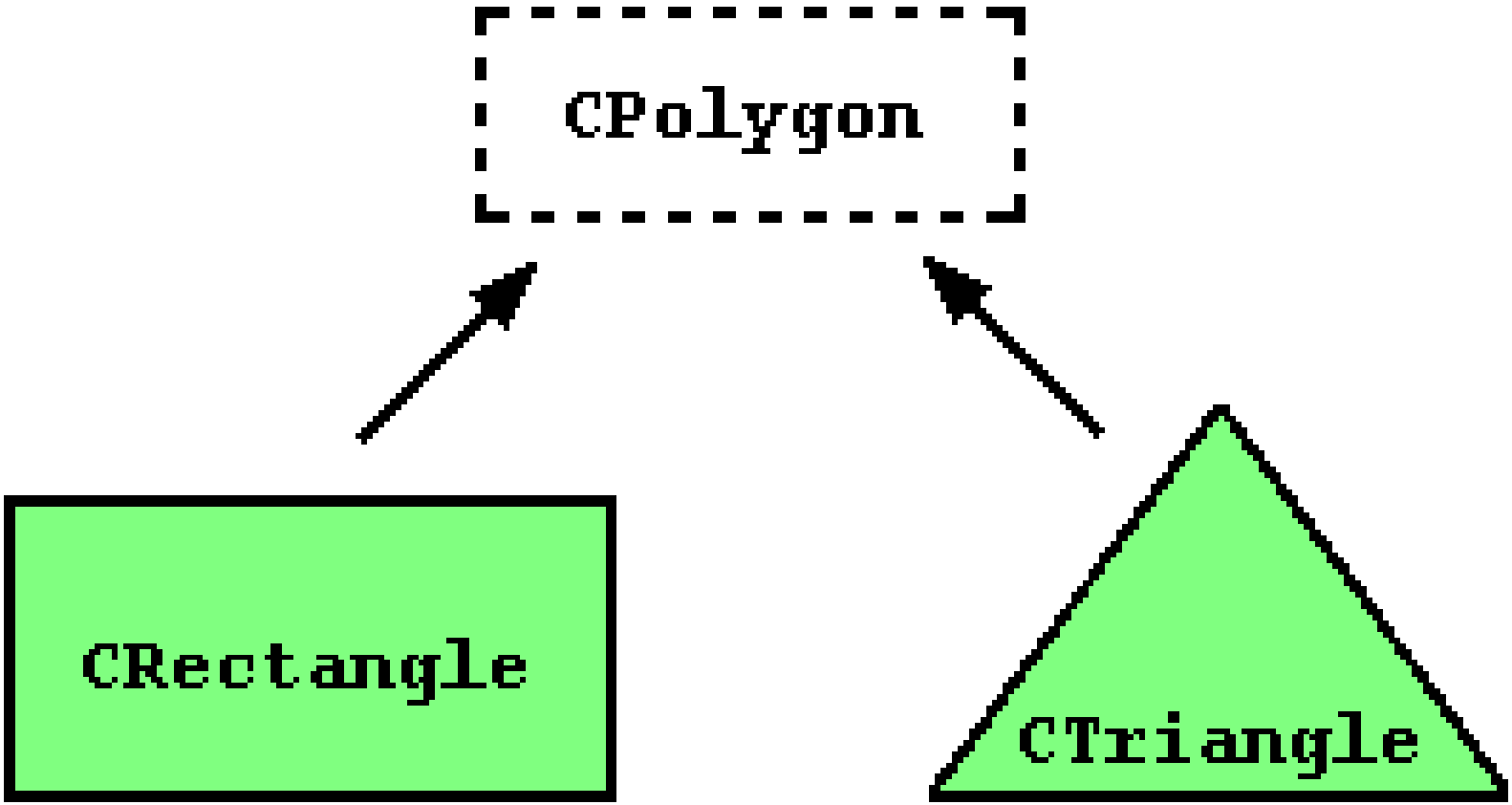


Multiple derived classes inherits from a single base class.



# Types of inheritance: Hierarchical





# Types of inheritance: Hierarchical

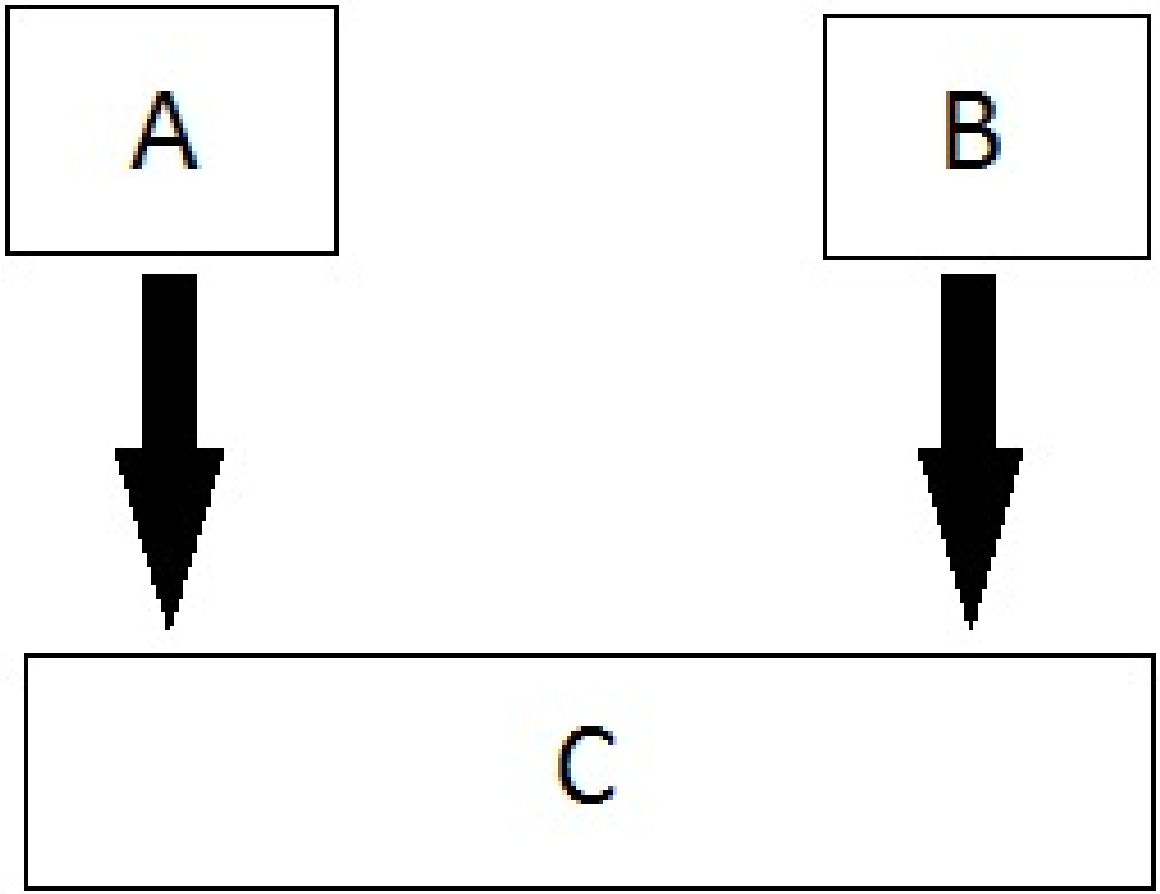


```
class Polygon {
  protected:
    int width, height;
  public:
    void set_values(int a, int b){
      width=a; height=b;
    }
};

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

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

int main() {
  Rectangle rect;
  Triangle trgl;
  rect.set_values(4,5);
  trgl.set_values(4,5);
  cout << rect.area() << endl; // 20
  cout << trgl.area() << endl; // 10
  return 0;
}
```

# Types of inheritance: Multiple



A single derived class may inherit from two or more than two base classes.



# Types of inheritance: Multiple

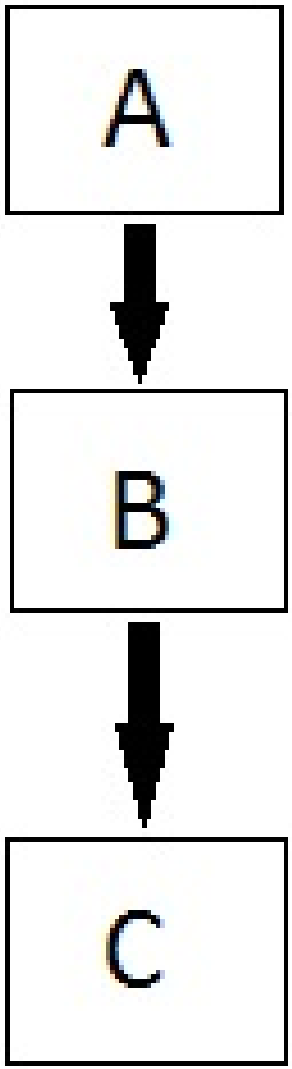


```
class Shape {
   public:
      void setWidth(int w) {width = w;}
      void setHeight(int h) {height = h;}

   protected:
      int width;
      int height;
};

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

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();
   cout << Rect.getArea() << endl;    // 35
   cout << Rect.getCost(area) << endl;    // 2450
   return 0;
}
```

# Types of inheritance: Multilevel



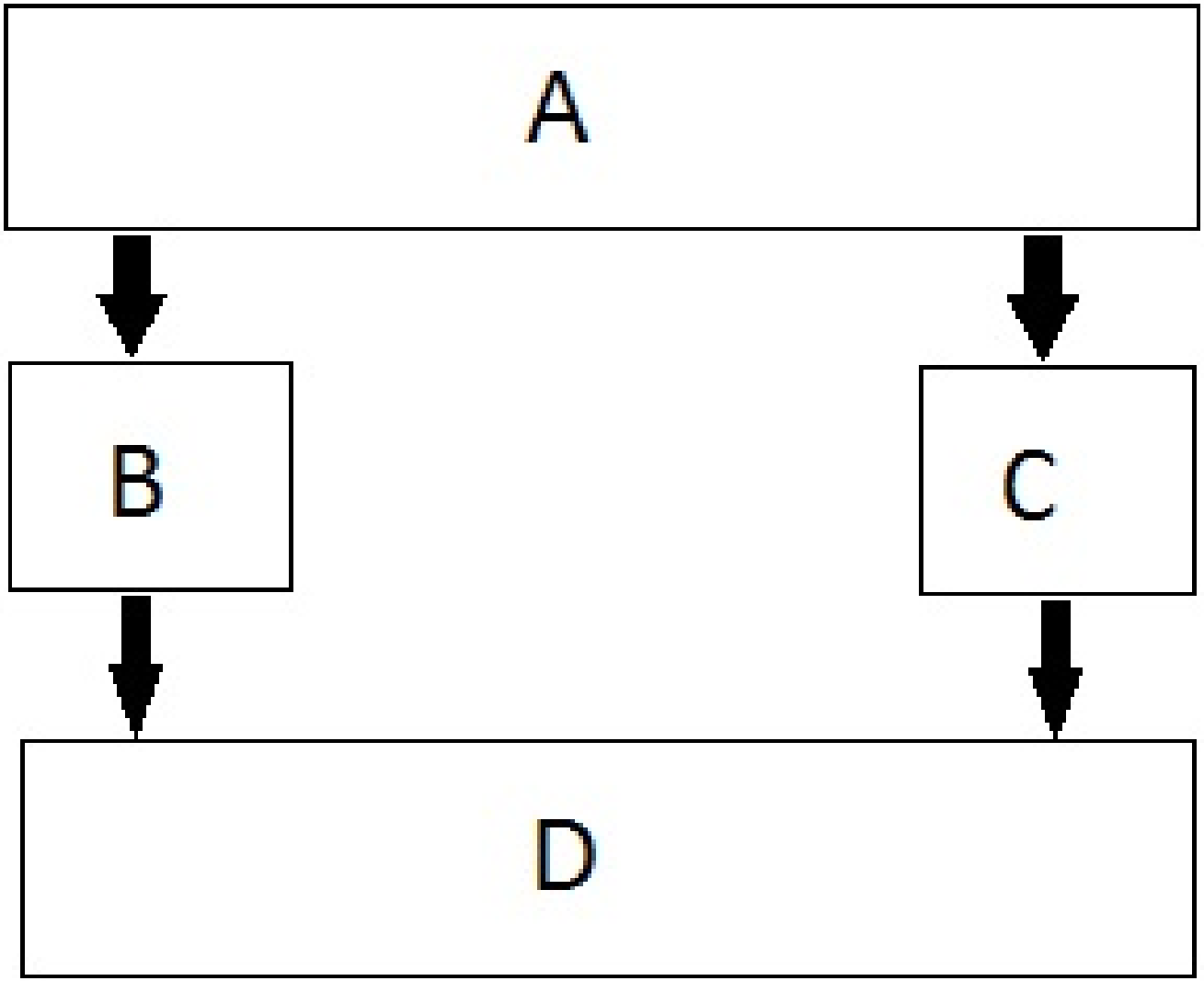
Derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.



# Types of inheritance: Hybrid



Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.



# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# What is not inherited from the base class?



- Constructors, copy constructor, and destructor of the base class
- Assignment operator=()
- Friends of the base class
- Originally private members of the base class (inherited but not accessible)

# Kahoot

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Constructors of base and derived classes



- Base class constructors are always called with the derived class constructors
- First the base class default constructor is executed, and then the derived class's constructor

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Constructor of base class is called with the derived



```
class Base {
  int x; // Remember, classes default to private
  public:
    Base() {cout << "Base default constructor";}
};

class Derived : public Base {
  int y;
  public:
    Derived(){cout << "Derived default constr";}
    Derived(int i){
      y = i;
      cout << "Derived parameterized";
    }
};

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

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Parameterized versus default



- Unless otherwise specified, the constructors of a derived class calls the default constructor of its base class(es)
- To call base class's parameterized constructor inside derived class's parameterized constructor, we must mention it explicitly while declaring derived class's parameterized constructor.

# Force calling of base parameterized constructor



```
class Base{
  public:
    int x;
    Base(int i){ x = i; }
};

class Derived: public Base {
  public:
    int y;

    // Specify base parameterized constructor
    Derived(int j): Base(j) { // Options here?
      y = j;
    }
};

int main(){
  Derived d(10);
  cout << d.x;     // 10
  cout << d.y;     // 10
}
```

# Outline




Introduction
Definition
Purpose
Example
Access control
public mode
protected mode
private mode
Inheritance
Types of inheritance
Single
Hierarchical
Multiple
Multilevel
Hybrid
Not inherited
Constructors
Constructor of base class
Parameterized versus default
Polymorphism and abstract classes

# Next time



Polymorphism and abstract classes