Inheritance

Lecture 18

Implicit Derived-Class Object to Base-Class Object Conversion

- `baseClassObject = derivedClassObject;`
  - This will work
    - Remember, the derived class object has more members than the base class object
  - Extra data is not given to the base class
- `derivedClassObject = baseClassObject;`
  - May not work properly
    - Unless an assignment operator is overloaded in the derived class, data members exclusive to the derived class will be unassigned
  - Base class has less data members than the derived class
    - Some data members missing in the derived class object

Contd..

- Four ways to mix base and derived class pointers and objects:
  - Referring to a base-class object with a base-class pointer
    - Allowed
  - Referring to a derived-class object with a derived-class pointer
    - Allowed
  - Referring to a derived-class object with a base-class pointer
    - Possible syntax error
    - Code can only refer to base-class members, or syntax error
  - Referring to a base-class object with a derived-class pointer
    - Syntax error
    - The derived-class pointer must first be cast to a base-class pointer

Composition vs. inheritance

- "is a" relationship
  - Inheritance
- "has a" relationship
  - Composition - class has an object from another class as a data member

```
Employee “is a” BirthDate;   //Wrong!
Employee “has a” BirthDate;//Composition
```

Virtual functions

Unit-6

- Polymorphism in inheritance hierarchies ??

Polymorphism

- Enables us to “program in the general” rather than “program in the specific”
- Programs that process objects of classes that are part of the same class hierarchy as if they are objects of the base class

Polymorphism

- One function can cause different actions to occur, depending on the type of the object on which the function is invoked

Static and Dynamic Binding

- When a reference to a *member* function is resolved at compile time, then static binding is used.
- When a reference to a member function can only be resolved at run-time, then this is called dynamic binding.

Polymorphism and Dynamic Binding

- To implement polymorphism, the programming language must support dynamic binding.
  - Polymorphism----- a concept
  - Dynamic binding -----implementation

Polymorphism and Dynamic Binding

- Classes rectangle, square is derived from class Quadrilateral
- area() is a member of all
  - But the way it is calculated is different
- Program invokes area() through quadrilateral class pointer, C++ dynamically chooses the correct function
  - polymorphism

Polymorphism

- Polymorphism facilitates adding new classes to a system with minimal modifications to its code

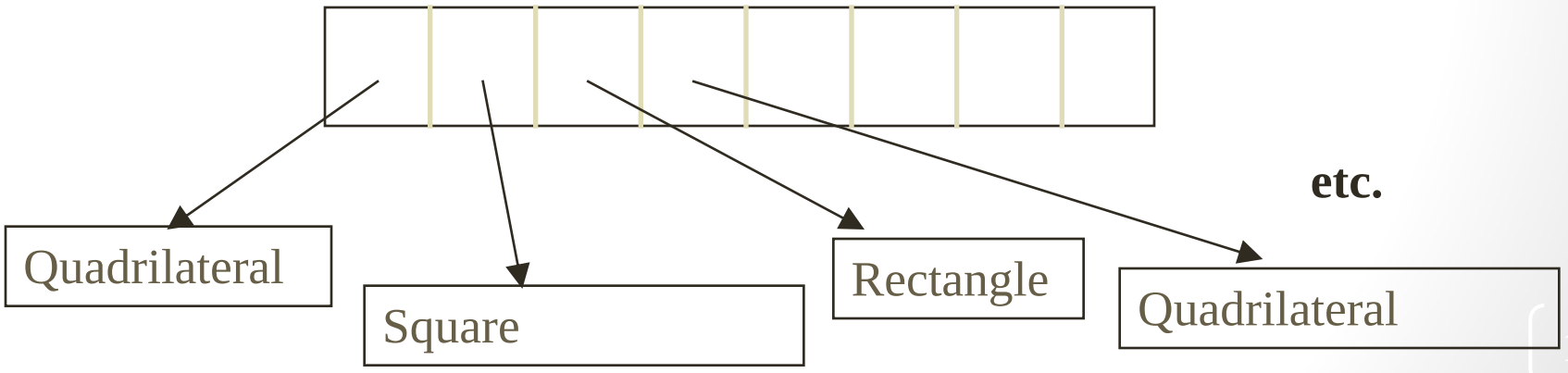
Upcasting

- An `Quadrilateral*` can hold a `square*`
- Because of the “is-a” relationship
- A **square** can take the place of an Employee object

- Hence, an array of `Quadrilateral*` can hold pointers to a mixture of the two types

The Goal

- To treat all objects as base objects
  - via a pointer-to-base
- But to have their behavior vary automatically
  - depending on the *dynamic type* of the object



- Through the quadrilateral class pointer (base class object) we call the function area(), and biding is done at run-time

Polymorphism in C++

- Virtual Function
  - A non-static member function prefaced by the *virtual* specifier.
  - It tells the compiler to generate code that selects the appropriate version of this function at run-time.

Example

```
class quadrilateral
{  public:
   virtual void area() { }; };
```

```
void main()
{   quadrilateral *q=new
    square(3);
   q->area();   delete q;
   q=new rectangle(2,4);
   q->area();}
```

```
class square : public quadrilateral
{  int side;
  public:  square(int i=1) { side=i; }
  void area() { cout<<"\n Area of square is : "<<(side*side); }};
```

```
class rectangle : public quadrilateral
{ int side1;  int side2;
  public:  rectangle(int i,int j) { side1=i; side2=j; }
  void area() { cout<<"\n Area of rectangle is : <<(2*side1*side2);}
```

Virtual function

- With virtual functions, the type of the object being pointed to, not the type of handle, determines which version of a virtual function to invoke.

- Dynamic binding

Dynamic vs. static binding

- When a virtual function is called by referencing a specific object by name (.), the binding is static

- Dynamic binding with virtual functions occurs only off pointer

Assignment

- Difference between Static and dynamic binding.