

# EECS402 Lecture 23

Andrew M. Morgan

Savitch Ch. 15 Polymorphism



#### Inheritance Revisited

402 402

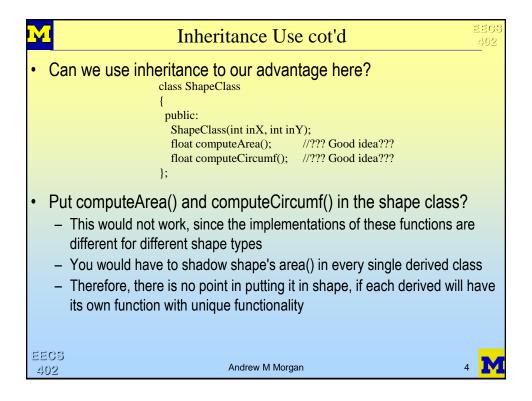
- Recall that inheritance allows one class to obtain attributes and functionality from another class
- Base classes should be generic-type classes which contain members common to all objects
- Many new classes can inherit from the base class
- A set of classes which all inherit from the same base class are related in that sense
- This relationship can be advantageous in C++

트ECS 402

Andrew M Morgan

M

```
Inheritance and Use
   Note the interfaces for the derived classes
                                class ShapeClass
                                 public:
                                  ShapeClass inX, int inY);
class SquareClass:public ShapeClass
                                         class RectangleClass:public ShapeClass
                                          public:
 public:
  SquareClass inX, int inY, int inSize);
                                           RectangleClass (int inX, int inY, int inLen, int inWid);
  float computeArea();
                                           float computeArea();
  float computeCircumf();
                                           float computeCircumf();
class CircleClass:public ShapeClass
                                         class RightTriangleClass:public ShapeClass
 public:
                                          public:
  CircleClass (int inX, int inY, int inRad);
                                           RightTriangleClass(int inX, int inY, int inLen, int inWid);
  float computeArea();
                                           float computeArea();
  float computeCircumf();
                                           float computeCircumf();
EECS
                                        Andrew M Morgan
 403
```



```
Discussion of Sample Program

Now we will consider a program dealing with shapes and areas, etc...

class ShapeClass
{
   public:
        ShapeClass(int inX, int inY):xPos(inX),yPos(inY)
        { }
        private:
        int xPos;
        int yPos;
        ShapeClass():xPos(0),yPos(0)
        { }
    };

EEGS
402

Andrew M Morgan

5
```

```
Program Without Polymorphism, p. 1
class SquareClass:public ShapeClass
   SquareClass(int inX, int inY, int inSize):ShapeClass(inX, inY), len(inSize)
   float computeArea()
     cout << "In square::area" << endl;</pre>
     res = (float) (len * len);
     return res;
   float computeCircumf()
     float res;
     res = (float)(4 * len);
     return res;
 private:
   SquareClass():ShapeClass(0, 0), len(1)
};
EECS
                                  Andrew M Morgan
402
```

```
Program Without Polymorphism, p. 2
lass RectangleClass:public ShapeClass
   RectangleClass(int inX, int inY, int inLen, int inWid):ShapeClass(inX, inY),
                                                len(inLen), wid(inWid)
  float computeArea()
    float res;
    cout << "In rectangle::area" << endl;</pre>
    res = (float)(len * wid);
    return res;
   float computeCircumf()
    float res;
    res = (float)(2 * len + 2 * wid);
    return res;
private:
  int len;
  int wid;
  RectangleClass():ShapeClass(0, 0), len(1), wid(1)
EECS
                                  Andrew M Morgan
402
```

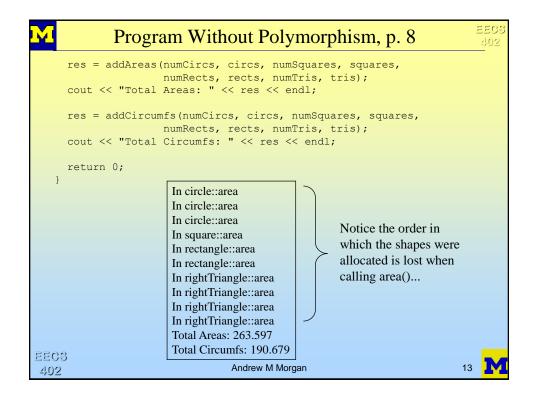
```
Program Without Polymorphism, p. 3
class CircleClass:public ShapeClass
  public:
    CircleClass(int inX, int inY, int inRad):ShapeClass(inX, inY), rad(inRad)
    float computeArea()
     cout << "In circle::area" << endl;</pre>
      res = M_PI * rad * rad;
     return res;
    float computeCircumf()
      float res;
      res = M_PI * 2 * rad;
     return res;
  private:
    int rad;
    CircleClass():ShapeClass(0, 0), rad(1)
};
EECS
                                  Andrew M Morgan
402
```

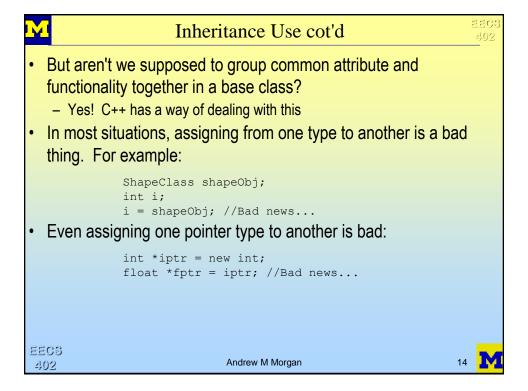
```
Program Without Polymorphism, p. 4
class RightTriangleClass:public ShapeClass
   RightTriangleClass(int inX, int inY, int inLen, int inWid):ShapeClass(inX, inY),
                                                              len(inLen), wid(inWid)
   float computeArea()
     float res;
     cout << "In rightTriangle::area" << endl;</pre>
     res = 0.5 * len * wid;
     return res;
   float computeCircumf()
     float res;
     res = len + wid + sqrt(len*len + wid*wid);
     return res;
 private:
   int len;
   int wid;
   RightTriangleClass():ShapeClass(0, 0), len(1), wid(1)
};
EECS
                                  Andrew M Morgan
403
```

```
Program Without Polymorphism, p. 5
       float addAreas(int numCircs, CircleClass **circs,
                      int numSquares, SquareClass **squares,
                      int numRects, RectangleClass **rects,
                      int numTris, RightTriangleClass **tris)
         int i;
         float res = 0.0;
         for (i = 0; i < numCircs; i++)
           res += circs[i]->computeArea();
         for (i = 0; i < numSquares; i++)
           res += squares[i]->computeArea();
         for (i = 0; i < numRects; i++)
           res += rects[i]->computeArea();
         for (i = 0; i < numTris; i++)</pre>
          res += tris[i]->computeArea();
         return res;
EECS
                                Andrew M Morgan
402
```

```
Program Without Polymorphism, p. 6
      float addCircumfs(int numCircs, CircleClass **circs,
                        int numSquares, SquareClass **squares,
                        int numRects, RectangleClass **rects,
                        int numTris, RightTriangleClass **tris)
        int i;
        float res = 0.0;
        for (i = 0; i < numCircs; i++)
          res += circs[i]->computeCircumf();
        for (i = 0; i < numSquares; i++)
          res += squares[i]->computeCircumf();
        for (i = 0; i < numRects; i++)
          res += rects[i]->computeCircumf();
        for (i = 0; i < numTris; i++)
          res += tris[i]->computeCircumf();
        return res;
EECS
                                Andrew M Morgan
703
```

```
Program Without Polymorphism, p. 7
 int main (void)
   float res;
   int i;
   int numCircs = 3, numSquares = 1;
   int numRects = 2, numTris = 4;
   //Declare arrays of pointers
   CircleClass **circs = new CircleClass*[numCircs];
   SquareClass **squares = new SquareClass*[numSquares];
   RectangleClass **rects = new RectangleClass*[numRects];
   RightTriangleClass **tris = new RightTriangleClass*[numTris];
   int c = 0, s = 0, r = 0, t = 0;
  circs[c++] = new CircleClass(0, 0, 2);
   rects[r++] = new RectangleClass(0, 0, 2, 2);
   rects[r++] = new RectangleClass(4, 6, 8, 8);
   tris[t++] = new RightTriangleClass(0, 0, 2, 2);
   squares[s++] = new SquareClass(3, 12, 6);
   tris[t++] = new RightTriangleClass(10, 10, 6, 8);
                                                                Notice the order in
  circs[c++] = new CircleClass(-10, -10, 4);
                                                                which the shapes are
  tris[t++] = new RightTriangleClass(0, 22, 4, 5);
   circs[c++] = new CircleClass(8, 3, 4);
                                                                allocated...
   tris[t++] = new RightTriangleClass(20, 8, 3, 7);
 continued on next page
EECS
                                   Andrew M Morgan
402
```







## Derived Ptr Assigned to Base Ptr

보면() 402

- However, there is one case where assigning different types is ok
- You are allowed to assign a pointer to a derived class to a variable declared as a pointer to the base class
  - This means you can do:

```
ShapeClass *shapePtr;
CircleClass *circPtr = new CircleClass(5, 5, 1);
RectangleClass *rectPtr = new RectangleClass(1, 1, 4, 4);
shapePtr = circPtr; //NOT bad news...
shapePtr = rectPtr; //NOT bad news...
```

- Logically, think of it this way:
  - A circle is a shape it is just a specialized shape
  - Therefore, if I am pointing to a shape, I might be pointing to a circle, or a square, or ...

EECS 402

Andrew M Morgan

M





## But Why?

402 402

- Obvious question: Why would I want to set a shape pointer to a circle pointer?
  - To allow polymorphism!
  - In the sample program discussed, we declared 4 separate arrays (square, circle, rectangle, triangle)
- Consider the code:

```
ShapeClass *shapePtr;
CircleClass *circPtr = new CircleClass(5, 5, 1);
shapePtr = circPtr;    //NOT bad news...
shapePtr->computeArea();    //Bad news...
```

- The above will not compile
  - It is looking for a member function of the shape class, called computeArea(), but one does not exist (check the ShapeClass definition!)

EECS 402

Andrew M Morgan

M



## Again - Then Why?

보03 보03

```
ShapeClass *shapePtr;
CircleClass *circPtr = new CircleClass(5, 5, 1);
shapePtr = circPtr;    //NOT bad news...
shapePtr->computeArea();    //Bad news...
```

- If we can't do this, then why bother with the assignment "sPtr = cPtr:"?
  - We're not done just yet we CAN do this, with the correct syntax
- The base class must be aware that a function called computeArea() exists
- The derived classes should have an implementation for the function computeArea() (for this example)
- The base class must be aware that the computeArea() function is special, in that derived classes may have overridden it

402 로로CS

Andrew M Morgan

17





## The Keyword virtual

트트C8 402

- The keyword "virtual" is used in C++ for this purpose
  - Functions can be declared as virtual in a class definition
  - Virtual functions are special member functions
  - When a pointer-to-an-object calls a virtual function:
    - First the type of the object being pointed to is determined either actually a pointer to an object of this class, or a pointer to an object of a derived class
    - · Second that type is checked to see if it has an implementation to the function
      - If so, the implementation from the derived class is used
      - If not, the implementation from the base class is used
- A virtual function does not need to be overridden, if you don't want different functionality from what is provided in the base class

EECS 402

Andrew M Morgan

1

```
Simple Example
    class W
                                            From main:
     public:
                                                                 W's print!
                                            W *Wptr;
       virtual void print()
                                            W Wobj;
                                                                 X's print!
        { cout << "W's print!" << endl; }
                                            X Xobj;
                                                                 W's print!
                                            Y Yobj;
   class X:public W
                                                                 Z's print!
                                            Z Zobj;
     public:
                                            Wobj.print();
        void print()
                                                                 W's print!
                                            Xobj.print();
        { cout << "X's print!" << endl; }
                                                                 X's print!
                                            Yobj.print();
                                            Zobj.print();
                                                                 W's print!
   class Y:public W
                                                                 Z's print!
                                            cout << endl
     public:
       void notprint()
                                            Wptr = &Wobj;
                                                              Assign to the W obj
        { cout << "Y's func!" << endl; }
                                            Wptr->print();
                                            Wptr = &Xobj;
   class Z:public W
                                            Wptr->print();
                                            Wptr = &Yobj;
     public:
                                            Wptr->print();
       void print()
                                            Wptr = &Zobj;
        { cout << "Z's print!" << endl; }
                                            Wptr->print();
    };
EECS
                                  Andrew M Morgan
403
```

```
New Discussion of Sample Program
          class ShapeClass
            public:
              ShapeClass(int inX, int inY):xPos(inX),yPos(inY)
              virtual float computeArea()
                cout << "No area implementation for this shape!" << endl;</pre>
                return 0.0;
              virtual float computeCircumf()
                cout << "No circumf implementation for this shape!" << endl;</pre>
                return 0.0;
            private:
              int xPos;
              int yPos;
              ShapeClass():xPos(0), yPos(0)
          };
EECS
                                 Andrew M Morgan
402
```



## New Derived Shape Classes

**芝三〇** 402

- There are no changes made to the SquareClass, RightTriangleClass, etc classes.
- The derived shape classes remain exactly as they were in the previous example
- Keyword virtual only needs to show up in the base class

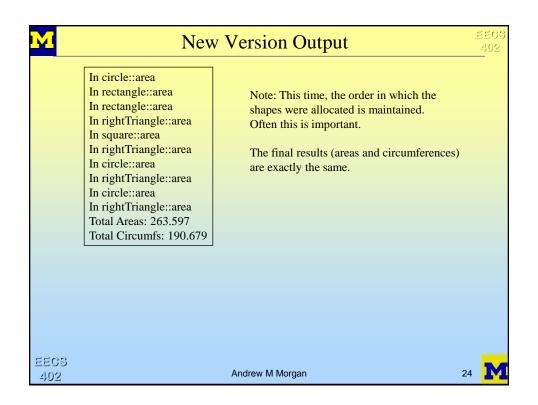
로르CS 402

Andrew M Morgan

M

```
New addAreas and addCircumfs Functions
float addAreas(int numShapes, ShapeClass **shapes)
  float res = 0.0;
                                                            Rather than pass in an
  for (i = 0; i < numShapes; i++)
                                                            array and count for each
    res += shapes[i]->computeArea();
                                                            type, just pass in one array
                                                            of shapes (since all types
                                                            are shapes via the "is-a"
  return res;
                                                            relationship inheritance
                                                            provides).
float addCircumfs(int numShapes, ShapeClass **shapes)
                                                            If adding a new shape type,
  int i;
                                                            these functions need not
  float res = 0.0;
 for (i = 0; i < numShapes; i++)
                                                            change in ANY way.
    res += shapes[i]->computeCircumf();
  return res;
EECS
                                   Andrew M Morgan
402
```

```
New main() Using Polymorphism
int main(void)
  float res;
  int i, numShapes, ctr = 0;
  int numCircs = 3, numSquares = 1;
                                                                   Rather than 4 separate
  int numRects = 2, numTris = 4;
                                                                   arrays of specific types, I
  numShapes = numCircs + numSquares + numRects + numTris;
                                                                   just declare 1 array of
   //Declare arrays of pointers
                                                                   type shape*.
  ShapeClass **shapes = new ShapeClass*[numShapes];
  shapes[ctr++] = new CircleClass(0, 0, 2);
                                                                   Since base class pointers
  shapes[ctr++] = new RectangleClass(0, 0, 2, 2);
                                                                   can be assigned derived
  shapes[ctr++] = new RectangleClass(4, 6, 8, 8);
  shapes[ctr++] = new RightTriangleClass(0, 0, 2, 2);
                                                                   class pointers, just store
  shapes[ctr++] = new SquareClass(3, 12, 6);
                                                                   all object pointers in the
  shapes[ctr++] = new RightTriangleClass(10, 10, 6, 8);
shapes[ctr++] = new CircleClass(-10, -10, 4);
                                                                   shape array.
  shapes[ctr++] = new RightTriangleClass(0, 22, 4, 5);
  shapes[ctr++] = new CircleClass(8, 3, 4);
  shapes[ctr++] = new RightTriangleClass(20, 8, 3, 7);
  res = addAreas(numShapes, shapes);
  cout << "Total Areas: " << res << endl;</pre>
  res = addCircumfs(numShapes, shapes);
  cout << "Total Circumfs: " << res << endl;</pre>
  return 0;
EECS
                                     Andrew M Morgan
703
```





## Reminder of Polymorphism

- The C++ construct of virtual functions allows one common interface for many implementations
- Virtual functions allow you to use the relationship between classes that derive from the same base class
- While each derived class may function differently, a common interface can exist
- Virtual functions have a body
  - It is executed when either:
    - 1. An object of the class type which defined the virtual function calls it
    - 2. A derived object pointer calls the function, but that object's class did not provide an implementation

EECS 703

Andrew M Morgan



```
More About Virtual Functions
 Consider this example, from a secretary's inventory control
  program:
                                         class PencilClass:public SupplyClass
class SupplyClass
                                          public:
public:
                                           void order()
 virtual void order()
                                           cout << "Call 555-6172 and "
   cout << "No order() defined!\n";</pre>
                                                << "ask for Sandy\n";
 void takeFromCabinet(int num)
                                         private:
                                           float leadSize;
    if (quantity >= num)
     quantity -= num;
                                         class InkPenClass:public SupplyClass
                                         private:
protected:
                                          int inkColor;
 string name; //name of supply
 int quantity; //# in cabinet now
EECS
                                Andrew M Morgan
402
```



#### The Problem...

보도**C**(

- So what is the problem with the previous program?
  - Some programmer who was in a hurry to meet a deadline forgot to override the order() function for the inkPen class
- So? It will call the order() that is defined in Supply right?
  - Yes, but that may not be acceptable.
  - In this situation, the base class definition of order() does not give any
    useful information of how to order the supply
  - Since your company might be an office inventory-control software company, this piece of software could be crucial to the survival of the company, and delivering bad programs could be a big problem for your job security
- How can I avoid this problem?
  - Make the function pure virtual!!

402 402

Andrew M Morgan

7





#### **Pure Virtual Functions**

트트C6

- A function that is declared pure virtual MUST be overridden in any class that derives from it
  - This is actually checked at compile time, so you ensure that any
    programmer that derives a new class from your base class will override it,
    or else it will not compile
- A function that is a pure virtual function need not (and can not) have a body defined
- But what if an object of the base class type tries to call that member function?
  - This can not happen!
  - When a class contains one or more pure virtual functions, it becomes an "abstract class"
  - C++ does not allow you to create objects from an abstract class
  - Abstract classes serve ONLY as base classes from which to inherit

EECS 402

Andrew M Morgan

28



