INTRODUCTION TO POLYMORPHISM

Dynamically choosing which function to call

Delroy A. Brinkerhoff

OBJECT-ORIENTED PARADIGM

- Encapsulation (Chap 9)
  - Data and operations packaged together
- Inheritance (Chaps 10)
  - Preeminent of five relationships
  - The subclass inherits all the features of the superclass
- Polymorphism (Chap 12)
  - Different objects respond to the same message differently
  - Delayed function binding

DRAWING SHAPES, PART 1

```
class Circle
{
    public:
        void draw();
};
```

```
class Rectangle
{
    public:
        void draw();
};
```

```
class Triangle
{
    public:
        void draw();
};
```

CHOOSING A SHAPE

```
cout << "C:\tCircle" << endl;
cout << "R:\tRectangle" << endl;
cout << "T:\tTriangle" << endl;

cout << "Please choose a shape: ";

char   choice;
cin >> choice;
cin.ignore();

Circle*       c;
Rectangle*    r;
Triangle*     t;
```

```
switch (choice)
{
        case 'C' :
        case 'c' :
                c = new Circle(…);
                break;

        case 'R' :
        case 'r' :
                r = new Rectangle(…);
                break;

        case 'T' :
        case 't' :
                t = new Triangle(…);
                break;
}
```

DRAWING (USING) SHAPES

```
switch (choice)
{
            case 'C' :
            case 'c' :
                        c->draw();
                        break;
            case 'R' :
            case 'r' :
                        r->draw();
                        break;
            case 'T' :
            case 't' :
                        t->draw();
                        break;
}
```

- What happens when we add a new shape?
  - Add a new pointer variable
  - Create a new class
  - Add a new case to instantiate an object
  - Add a new case <u>everywhere</u> we need to draw the shapes
- Polymorphism provides a more elegant solution

POLYMORPHISM REQUIREMENTS

- Inheritance
- Up casting
- A pointer or reference variable (polymorphism cannot operate through an automatic variable)
- Function overriding
- Virtual functions