ECE 220 Computer Systems & Programming

Lecture 24 – Intro to C++: Inheritance, Abstraction & Polymorphism

November 30, 2017

- Midterm 2 regrade due in labs this Friday
- Final exam conflict sign-up available now
- Quiz 6 next week

ECE ILLINOIS

ILLINOIS

Operator Overloading

Redefine built-in operators like +, -, *, <, >, = in C++ to do what you want

```
Example:
class vector {
    Protected:
    double angle, length;
    public:
    //constructors & other member functions
    …
    vector operator +(const vector &b) {
        vector c;
        double ax = length*cos(angle);
        double bx = b.length*cos(b.angle);
        double ay = length*sin(angle);
        double by = b.length*sin(b.angle);
        double cx = ax+bx;
        double cy = ay+by;
        c.length = sqrt(cx*cx+cy*cy);
        c.angle = acos( cx/c.length );
        return c;}
};
```

```
vector c(1.5,2);
vector d(2.6,3);

//before operator overload
vector e = c.add(d);

//after operator overload
vector e = c + d;
```

Inheritance & Abstraction

C++ allows us to define a class based on an existing class, and the new class will inherit members of the existing class.

- the **existing** class –
- the **new** class –

A derived class inherits all base class member functions with the following exceptions:

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

```
class orthovector : public vector{
    protected:
    int d; //direction can be 0,1,2,3, indicating r, l, u, d
    public:
    orthovector(int dir, double l){
         const double halfPI = 1.507963268;
         d = dir;
         angle = d*halfPI;
         length = l;
    }
    orthovector() {d = 0; angle = 0.0; length = 0.0;}
    double hypotenuse(orthovector b){
         if((d+b.d)%2 == 0) return length + b.length;
         return (sqrt(length*length + b.length*b.length));
    }
};
```

Access	public	protected	private
Same Class	Y	Y	Y
Derived Class	Y	Y	N
Outside Class	Y	N	N

Polymorphism

- a call to a member function will cause a **different function to be executed** depending on the type of the object that invokes the function

```
int main(){
    Rectangle rec(3,5);
    Triangle tri(4,5);

    rect.area();
    tri.area();

    return 0;
}
```

Example:

```
//base class
class Shape{
    protected:
    double width, height;
    public:
    Shape() {width = 1; height = 1;}
    Shape(double a, double b) { width = a; height = b; }
    double area() { cout << "Base class area unknown" << endl;
                    return 0; }
};
```

```
//derived classes
class Rectangle : public Shape{
    public:
    Rectangle(double a, double b) : Shape(a,b){}
    double area() {


    }
};

class Triangle : public Shape{
    public:
    Triangle(double a, double b) : Shape(a,b){}
    double area() {


    }
};
```

Declared Type vs. Actual Type

```
int main(){
        Shape *ptr;
        Rectangle rec(3,5);
        Triangle tri(4,5);

        //use ptr to point to rec object
        ptr = &rec;
        ptr->area();

        //use ptr to point to tri object
        ptr = &tri;
        ptr->area();

        return 0;
}
```

What does this program print?

Virtual Function

- **virtual functions** are member functions in the base class you expect to redefine in the derived classes
- derived class declares instances of that member function

```
//base class
class Shape{
    protected:
    double width, height;
    public:
    Shape() {width = 1; height = 1;}
    Shape(double a, double b) { width = a; height = b; }


};
```