ECE 220 Computer Systems & Programming

Inheritance, Polymorphism, Virtual Function

ECE ILLINOIS

ILLINOIS

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 – base class
- the **new** class – derived 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() {


    }
};
```

Base Class & Derived Class

```
//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 class
class Rectangle : public Shape{
    public:
    Rectangle(double a, double b) : Shape(a,b){}
    double area() {
    cout << "Rectangle object area is " << width*height << endl;
    return (double)width*height;  }
};
```

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();

        return 0;
}
```

What would 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

```
class Shape{
    protected:
    int width, height;
    public:
    Shape(int a, int b) { width = a; height = b; }
    virtual int area() { cout << "Base class area." << endl; return 0; }
};
class Rectangle : public Shape{
    public:
    Rectangle(int a, int b) : Shape(a,b){}
    int area() {
        cout << "Rectangle class area." << endl;
        return width*height; }
};
```

Virtual Function Table (VTable)

- stores pointers to all virtual functions
- created for each class that uses virtual functions
- lookup during the function call

Abstract Base Class & Pure Virtual Functions

```
class Shape{
    protected:
    int width, height;
    public:
    Shape(int a, int b) { width = a; height = b; }
    virtual int area()=0; //pure virtual function – it has no body
};
```

```
int main(){
    Shape shape1(2,4); // this will cause compiler error!
    Shape *p_shape1; // this is allowed
}
```

- derived class must define a body for this virtual function, or it will also be considered an abstract base class