UEE1303 S18: Object-Oriented Programming

LAB #11: Inheritance (II)

What you will learn from Lab 11

In this laboratory session you will: learn how to use multiple inheritance.

LAB11-1: Access to Base Classes

In the following example, B is a public base for X. Please fix the compiler error here.

```
//lab11-1.cpp
#include <iostream>
class B
{
public:
	B() { i = 0; f = 0.0; d = 0.0; }
	double d;
	void g1(B b){ f = b.f; }
protected:
	float f;
private:
	int i;
};
class X: public B
{
public:
	X() { s = 0; }
	void g2(X x) { f = x.f; }
	void g3(B b) { f = b.f; }
protected:
	short s;
};
int main()
{
	B b1;
	X x1;
x1.g1(b1);
	return 0;
}
```

- Please modify B as a protected base and compiler the program again.
- Here provide guidelines for access control:
  1. If B is a private base, its public and protected members become private members of derived class.
  2. If B is a protected base, its public and protected members become protected members of derived class.
  3. If B is a public base, its public members become members of derived class and its protected members become protected members of derived class.
- The access control for protected member, similar to private member, is that only its member and friend can access it. However, the protected member can become private or

protected members of derived class but private member cannot. Therefore, protected members of a class are designed for use by derived classes and are not intended for general use.

LAB11-2: Multiple Inheritances

1. A class can be directly derived from two or more base classes. This is called multiple inheritance. The class `Circle_in_Triangle` is derived from classes `Circle` and `Triangle`.

```
// lab11-2
#include <iostream>
using namespace std;
class Point2D
{
public:
	Point2D(){ x = 0; y = 0; }
	void display() const;
	// ...
private:
	int x;
	int y;
};
class Circle
{
public:
	void draw();
	// ...
private:
	Point2D center;
	double radius;
};
class Triangle
{
public:
	// ...
	~Triangle(){ delete [] vertices; }
	void draw();
private:
	Point2D *vertices;
};

class Circle_in_Triangle: public Circle, public Triangle
{
public:
	// ...
	void draw()
	{
		Circle::draw();
		Triangle::draw();
	}
};
int main()
{
	Point2D p;
```

```
    Point2D *vec = new Point2D [3];
    Circle_in_Triangle ct(p, 0, vec);
    ct.draw();
    return 0;
}
```

LAB11-3: Ambiguity Resolution

✓ When two base classes have members with the same name, they can be resolved by using the scope resolution operator.

```
// lab11-3.cpp
/* add area() for class Circle */
/* add area() for class Triangle */

int main()
{
    Point2D p;
    Point2D *vec = new Point2D [3];
    Circle_in_Triangle ct(p, 0, vec);
    ct.draw();
    cout << "Area of Circle: " << ct.Circle::area() << endl;
    cout << "Area of Triangle: " << ct.Triangle::area() << endl;
    cout << "Area of Circle_in_Triangle: " << ct.area() << endl;
    return 0;
}
```

- The compiler shows the error message “request for member `area' is ambiguous” on screen.
- A using-declaration can bring different functions from base classes to a derived class and then overload resolution can be applied. You can add “using Triangle::area;” in Circle_in_Triangle and compiler the program again.

LAB11-4: Replicated Base Classes

✓ With the possibility of derivation from two bases, a class can be a base twice.

```
// lab11-4.cpp
class Shape
{
protected:
    int color;
};
class Circle: public Shape
{
    // definition in lab9-3
};
class Triangle: public Shape
{
    // definition in lab9-3
};
class Circle_in_Triangle: public Circle, public Triangle
{
public:
    // ...
    void draw()
    {
```

```
        cout << "Circle's color: " << Circle::color << endl;
        cout << "Triangle's color: " << Triangle::color << endl;
        Circle::draw();
        Triangle::draw();
    }
};
```

- In this example, the colors of a Circle and a Triangle for an object of Circle_in_Triangle can be different.
- A using-declaration can bring different functions from base classes to a derived class and then overload resolution can be applied. You can add “using Triangle::area;” in Circle_in_Triangle and compiler the program again.

LAB11-5: Virtual Base Classes

✓ Often a base class need not be replicated. That is, only one copy of a replicated class need be inherited for a derived class object.

```
// lab11-5.cpp
class Shape
{
protected:
    int color;
};
class Circle: virtual public Shape
{
    // definition in lab10-3
};
class Triangle: virtual public Shape
{
    // definition in lab10-3
};
class Circle_in_Triangle: public Circle, public Triangle
{
public:
    // ...
    void draw()
    {
        cout << "Circle's color: " << Circle::color << endl;
        cout << "Triangle's color: " << Triangle::color << endl;
        Circle::draw();
        Triangle::draw();
    }
};
```

- In this example, the colors of a Circle and a Triangle for an object of **Circle_in_Triangle** are the same since they are inherited for the same base.

Exercise 11-1:

✓ Task A: Please finish the program lab11-2 and show the follows on screen.

```
Center: 0,0
Radius: 0
Vertices:
0,0
```

```
0,0
0,0
```

- ✓ Task B: Please finish the program lab11-3 and show the area information on screen. Note that you need to define the center and radius of circle, and the vertices of triangle on the main function. Moreover, the area of **Circuit_in_Triangle** is defined as (area in Triangle – area of Circle)

```
Center: 2,2
Radius: 1
Vertices:
1,1
3,1
1,6
Area of Circle: 3.14
Area of Triangle: 5.00
Area of Circuit_in_Triangle: 1.36
```

- ✓ Task C: Please finish the program lab11-5 and show the color information on screen

```
Circle's color: 255
Triangle's color: 255
Center: 2,2
Radius: 1
Vertices:
1,1
3,1
1,6
Area of Circle: 3.14
Area of Triangle: 5.00
Area of Circuit_in_Triangle: 1.36
```