**Polymorphism**

Let’s say there are three classes: CPolygon, CRectangle, and CTriangle

Assume **CPolygon** is the base class ( Parent )

**CRectangle** and **CTriangle** are derive class ( children )

class CPolygon

{

protected:

int width, height;

public:

void setup (int first, int second)

{

width= first;

height= second;

}

};

class CRectangle: public CPolygon

{

public:

int area()

{

return (width \* height);

}

};

class CTriangle: public CPolygon

{

public:

int area()

{

return (width \* height / 2);

}

};

Obviously the children classes inherit the width, height variables and setup function from the parent. They also have their own function called area, which can return the area size as an integer value. So every time you want to create a CRectangle object or a CTriangle object, you have to do like this:

int main ()

{

CRectangle rectangle;

CTriangle triangle;

rectangle.setup(2,2);

triangle .setup(2,2);

cout << rectangle.area () << endl;

cout << triangle.area () << endl;

return 0;

}

By this main file, you know there are two different object by the declaration:

**CRectangle rectangle;**

**CTriangle triangle**;

But for other readers, nobody can see the relation between these two object and their parent

So we can do another way.

int main ()

{

CRectangle rectangle;

CTriangle triangle;

CPolygon \* ptr\_polygon1 = &rectangle;

CPolygon \* ptr\_polygon2 = &triangle;

ptr\_polygon1->setup(2,2);

ptr\_polygon2->setup(2,2);

cout << rectangle.area () << endl;

cout << triangle.area () << endl;

return 0;

}

you create two CPolygon type pointers and let them point to the two objects’ address.

Now you can still use setup function by the pointers because the setup function is inherited from CPolygon.

**CPolygon \* ptr\_polygon1 = &rectangle;**

**CPolygon \* ptr\_polygon2 = &triangle;**

But you will find out that you can’t use area function by the pointers like this:

**cout << ptr\_polygon1->area () << endl;**

**cout << ptr\_polygon2->area () << endl;**

Because area function didn’t define in the parent class, but your pointers are CPolygon type pointers. So now we need to make a little changes for these three class.

If you want to design a very clear and complete relation in this case, You can define the area function also in the parent(base) class. Since the area function will do different calculation in different child class ( the formulas are different ), you can’t define the body in the parent. So you can add a ‘virtual’ keyword in from of the area function. Then let the function do nothing in the parent class.

By adding the ‘virtual’ keyword, the compiler will smart enough to know you actually define the body of area function in the children classes. So if you use a parent type pointer to point to the children type object. You can directly use the pointer to go through the virtual function, the compiler will know it’s a virtual function and try to find the same function name in the corresponding children class and use that one.

It will look like this:

class CPolygon

{

protected:

int width, height;

public:

void setup (int first, int second)

{

width= first;

height= second;

}

virtual int area()

{

return (0);

}

}

**Virtual destructor**

Check the result for both add “virtual” in front of the base class destructor and don’t add “virtual” then you will see the different.

class CPolygon

{

protected:

int width, height;

public:

void setup (int first, int second)

{

width= first;

height= second;

}

virtual int area() = 0;

**virtual** ~CPolygon()

{

cout << "destructor from CPolygon" << endl;

}

};

class CRectangle: public CPolygon

{

public:

int area()

{

return (width \* height);

}

~CRectangle()

{

cout << "destructor from CRectangle" << endl;

}

};

class CTriangle: public CPolygon

{

public:

int area()

{

return (width \* height / 2);

}

~CTriangle()

{

cout << "destructor from CTriangle" << endl;

}

};

int main(int argc, const char \* argv[]) {

CPolygon \* p1 = new CRectangle;

CPolygon \* p2 = new CTriangle;

p1->setup(2,2);

p2->setup(2,2);

cout << p1->area() << endl;

cout << p2->area() << endl;

delete p1;

delete p2;

return 0;

}