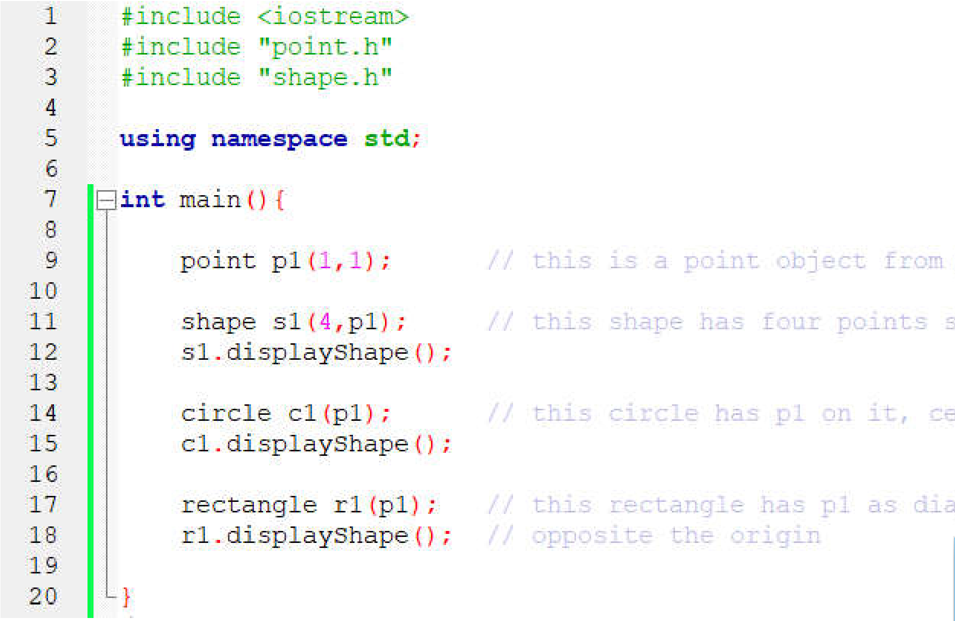
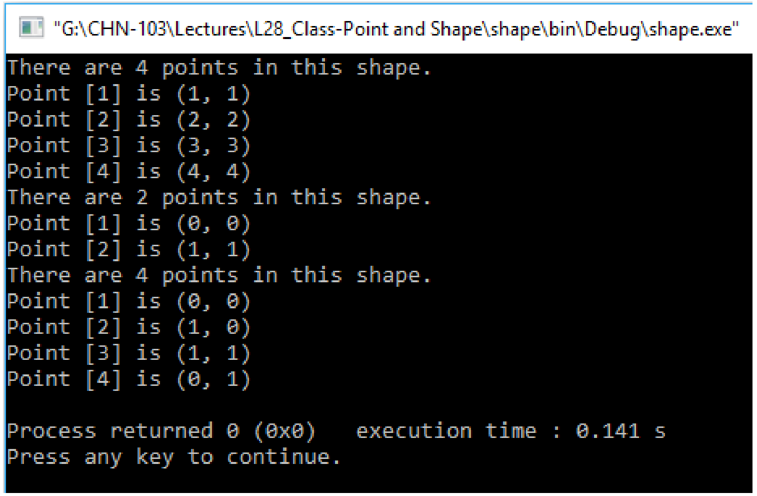
Lecture-29

Class – Polymorphism: *virtual* function *getArea()*

What is not polymorphism?



```
#include <iostream>
#include "point.h"
#include "shape.h"

using namespace std;

int main(){

    point p1(1,1);      // this is a point object from point class

    shape s1(4,p1);     // this shape has four points starting with p1
    s1.displayShape();

    circle c1(p1);      // this circle has p1 on it, center at (0,0)
    c1.displayShape();

    rectangle r1(p1);   // this rectangle has p1 as diagonal point
    r1.displayShape();  // opposite the origin

}
```



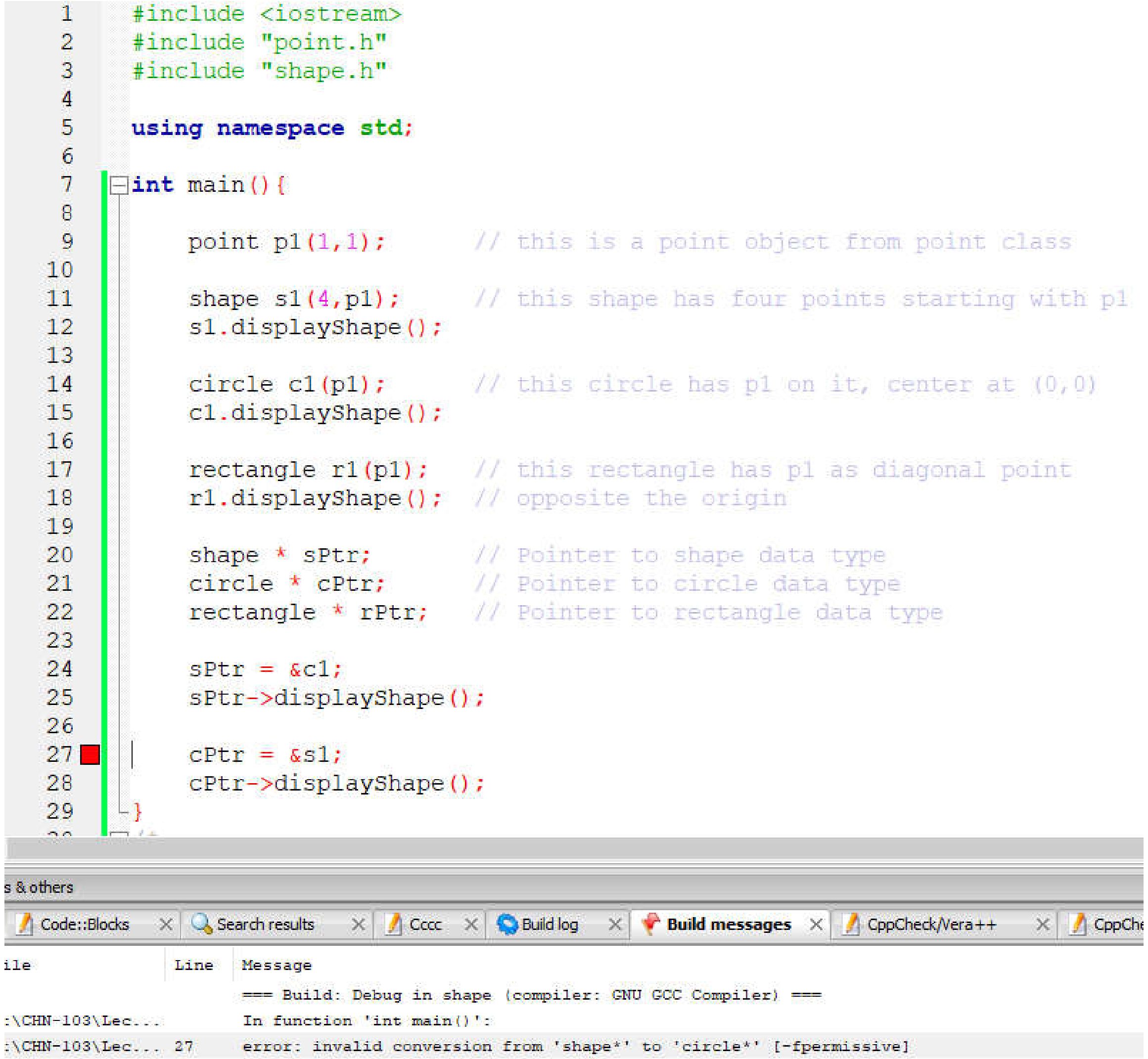
"G:\CHN-103\Lectures\L28_Class-Point and Shape\shape\bin\Debug\shape.exe"

```
There are 4 points in this shape.
Point [1] is (1, 1)
Point [2] is (2, 2)
Point [3] is (3, 3)
Point [4] is (4, 4)
There are 2 points in this shape.
Point [1] is (0, 0)
Point [2] is (1, 1)
There are 4 points in this shape.
Point [1] is (0, 0)
Point [2] is (1, 0)
Point [3] is (1, 1)
Point [4] is (0, 1)

Process returned 0 (0x0)   execution time : 0.141 s
Press any key to continue.
```

This is an example of code reuse made possible by inheritance, all objects s1, c1 and r1 use the same definition of *displayShape()* to output the result.

In polymorphism, we would like to execute different functions for different objects.

What is not polymorphism?

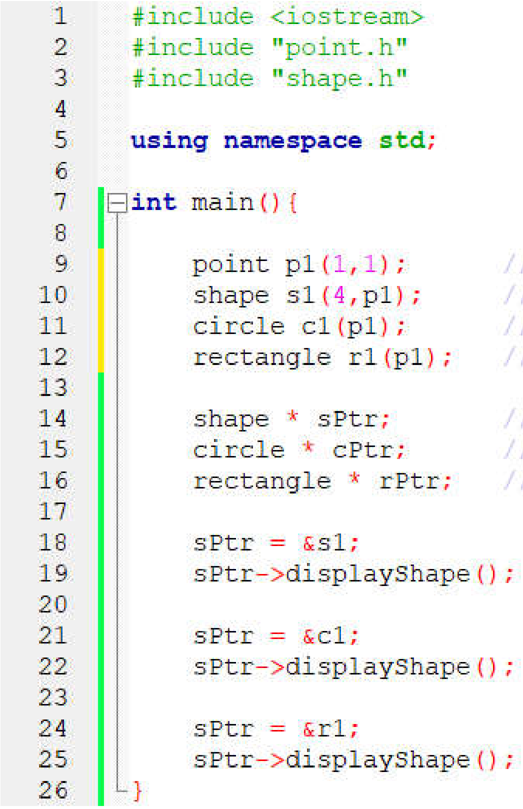


```cpp
#include <iostream>
#include "point.h"
#include "shape.h"

using namespace std;

int main(){

    point p1(1,1);       // this is a point object from point class

    shape s1(4,p1);      // this shape has four points starting with p1
    s1.displayShape();

    circle c1(p1);       // this circle has p1 on it, center at (0,0)
    c1.displayShape();

    rectangle r1(p1);    // this rectangle has p1 as diagonal point
    r1.displayShape();   // opposite the origin

    shape * sPtr;        // Pointer to shape data type
    circle * cPtr;       // Pointer to circle data type
    rectangle * rPtr;    // Pointer to rectangle data type

    sPtr = &c1;
    sPtr->displayShape();

    cPtr = &s1;
    cPtr->displayShape();
}
```

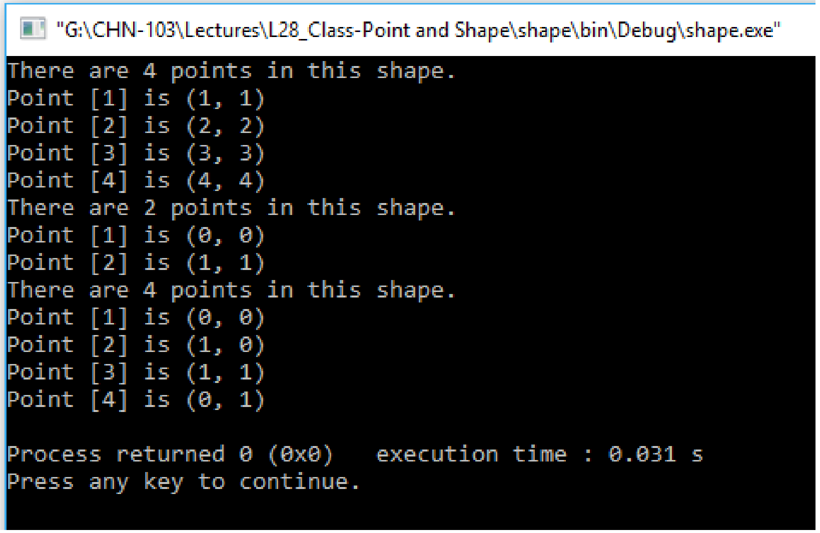
```
ile                 Line   Message
                           === Build: Debug in shape (compiler: GNU GCC Compiler) ===
:\CHN-103\Lec...           In function 'int main()':
:\CHN-103\Lec...    27     error: invalid conversion from 'shape*' to 'circle*' [-fpermissive]
```

We can use pointer to base class datatype (sPtr) to point at derived class object (c1), however the reverse (cPtr) cannot point at (s1).

What is not polymorphism?



```cpp
#include <iostream>
#include "point.h"
#include "shape.h"

using namespace std;

int main(){

    point p1(1,1);      // this is a point object from point class
    shape s1(4,p1);     // this shape has four points starting with p1
    circle c1(p1);      // this circle has p1 on it, center at (0,0)
    rectangle r1(p1);   // this rectangle has p1 as diagonal point

    shape * sPtr;       // Pointer to shape data type
    circle * cPtr;      // Pointer to circle data type
    rectangle * rPtr;   // Pointer to rectangle data type

    sPtr = &s1;
    sPtr->displayShape();

    sPtr = &c1;
    sPtr->displayShape();

    sPtr = &r1;
    sPtr->displayShape();
}
```

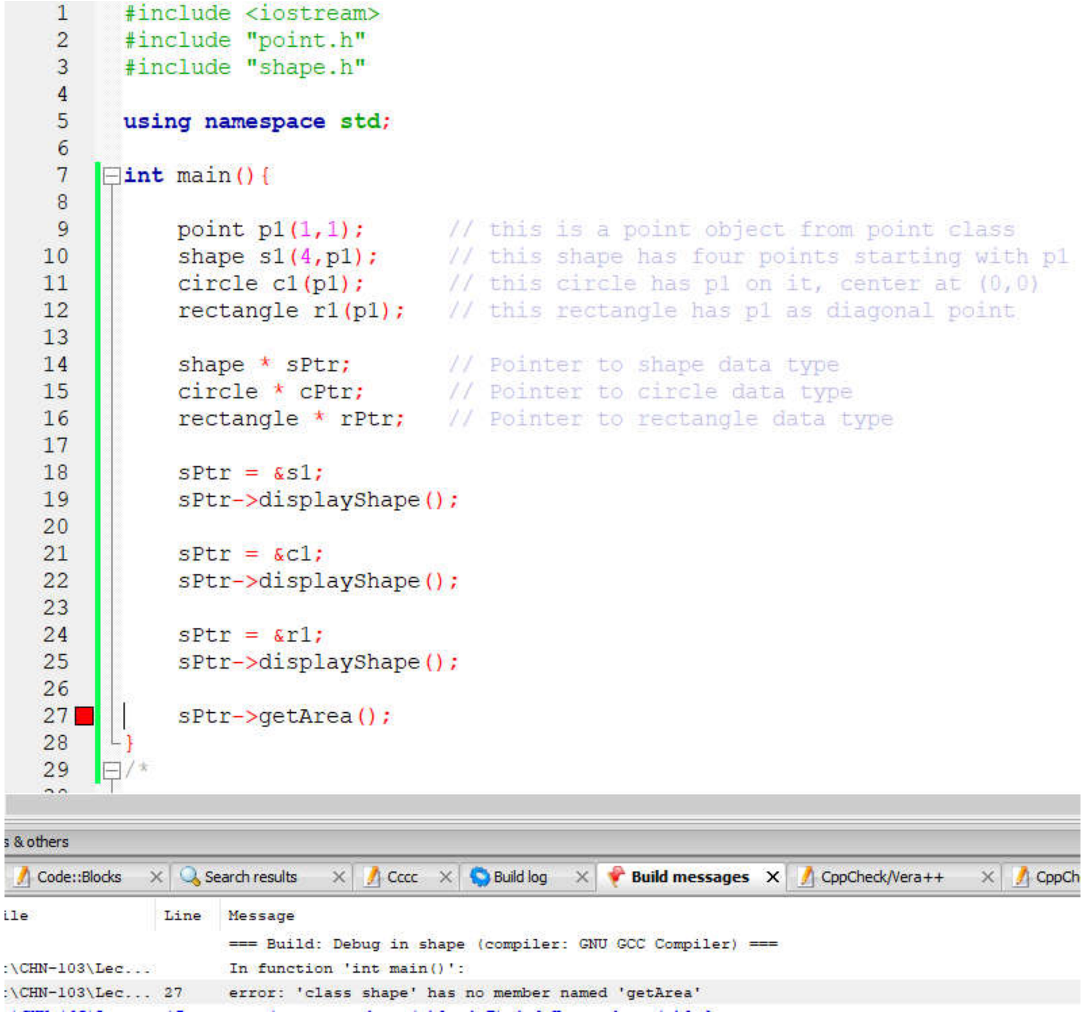


"G:\CHN-103\Lectures\L28_Class-Point and Shape\shape\bin\Debug\shape.exe"

```
There are 4 points in this shape.
Point [1] is (1, 1)
Point [2] is (2, 2)
Point [3] is (3, 3)
Point [4] is (4, 4)
There are 2 points in this shape.
Point [1] is (0, 0)
Point [2] is (1, 1)
There are 4 points in this shape.
Point [1] is (0, 0)
Point [2] is (1, 0)
Point [3] is (1, 1)
Point [4] is (0, 1)

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

The base class pointer can be used to execute the member functions that were part of base class and inherited into derived class.

What is not polymorphism?

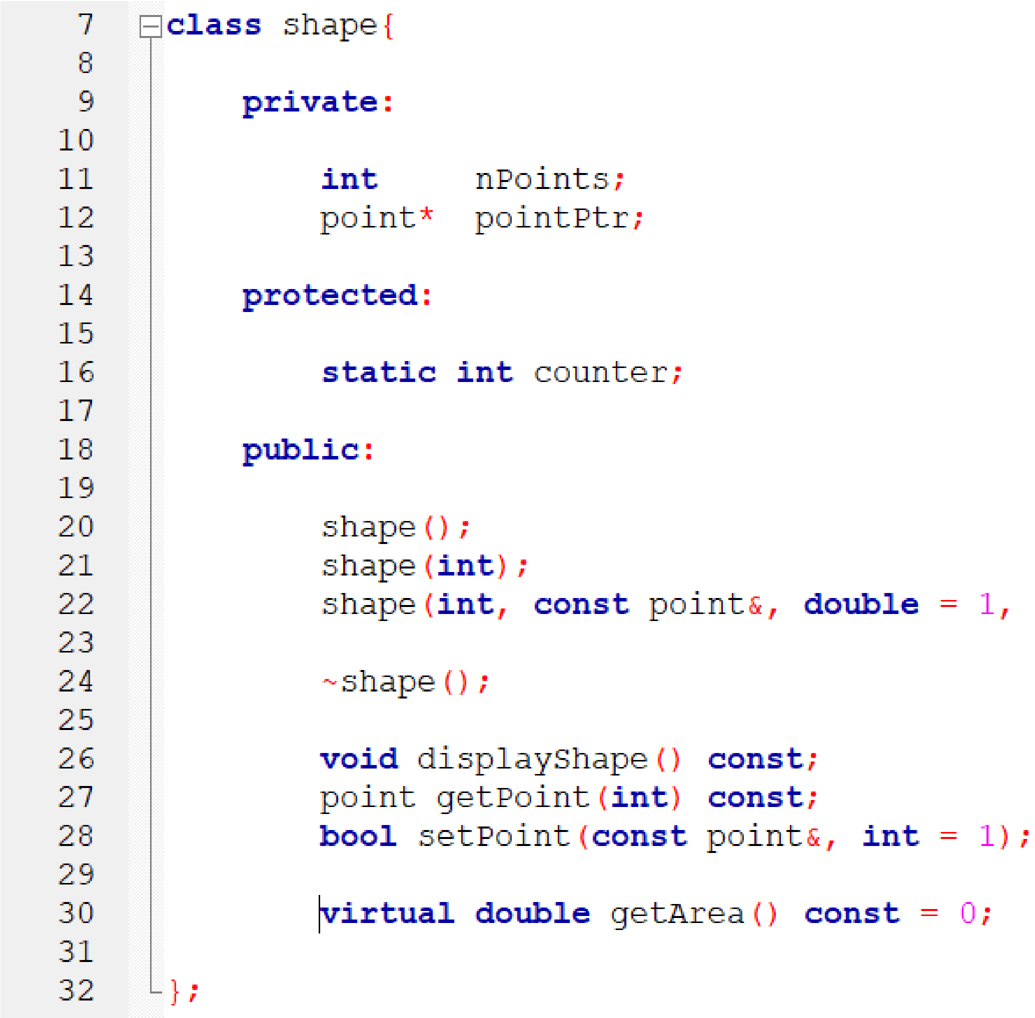


```
#include <iostream>
#include "point.h"
#include "shape.h"

using namespace std;

int main(){

    point p1(1,1);      // this is a point object from point class
    shape s1(4,p1);     // this shape has four points starting with p1
    circle c1(p1);      // this circle has p1 on it, center at (0,0)
    rectangle r1(p1);   // this rectangle has p1 as diagonal point

    shape * sPtr;       // Pointer to shape data type
    circle * cPtr;      // Pointer to circle data type
    rectangle * rPtr;   // Pointer to rectangle data type

    sPtr = &s1;
    sPtr->displayShape();

    sPtr = &c1;
    sPtr->displayShape();

    sPtr = &r1;
    sPtr->displayShape();

    sPtr->getArea();
}
/*
```

```
ile                 Line   Message
                           === Build: Debug in shape (compiler: GNU GCC Compiler) ===
:\CHN-103\Lec...           In function 'int main()':
:\CHN-103\Lec...    27     error: 'class shape' has no member named 'getArea'
```

The base class pointer cannot be used to execute the member functions that are part of derived class even though the pointer can point at a derived class object.

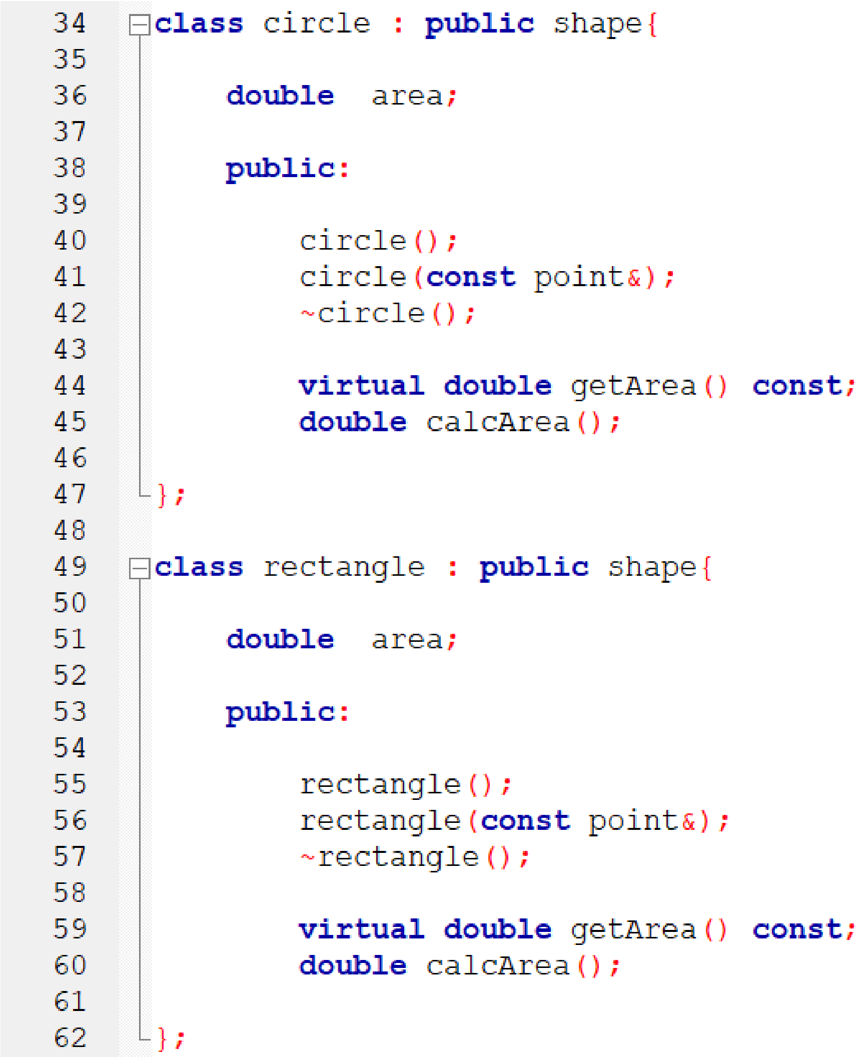
Polymorphism by *virtual* function



```
class shape{

    private:

        int     nPoints;
        point*  pointPtr;

    protected:

        static int counter;

    public:

        shape();
        shape(int);
        shape(int, const point&, double = 1, double = 1);

        ~shape();

        void displayShape() const;
        point getPoint(int) const;
        bool setPoint(const point&, int = 1);

        virtual double getArea() const = 0;

};
```

We make *getArea()* a *pure virtual* function of shape class. *Pure* because shape class does not provide an implementation of the function.

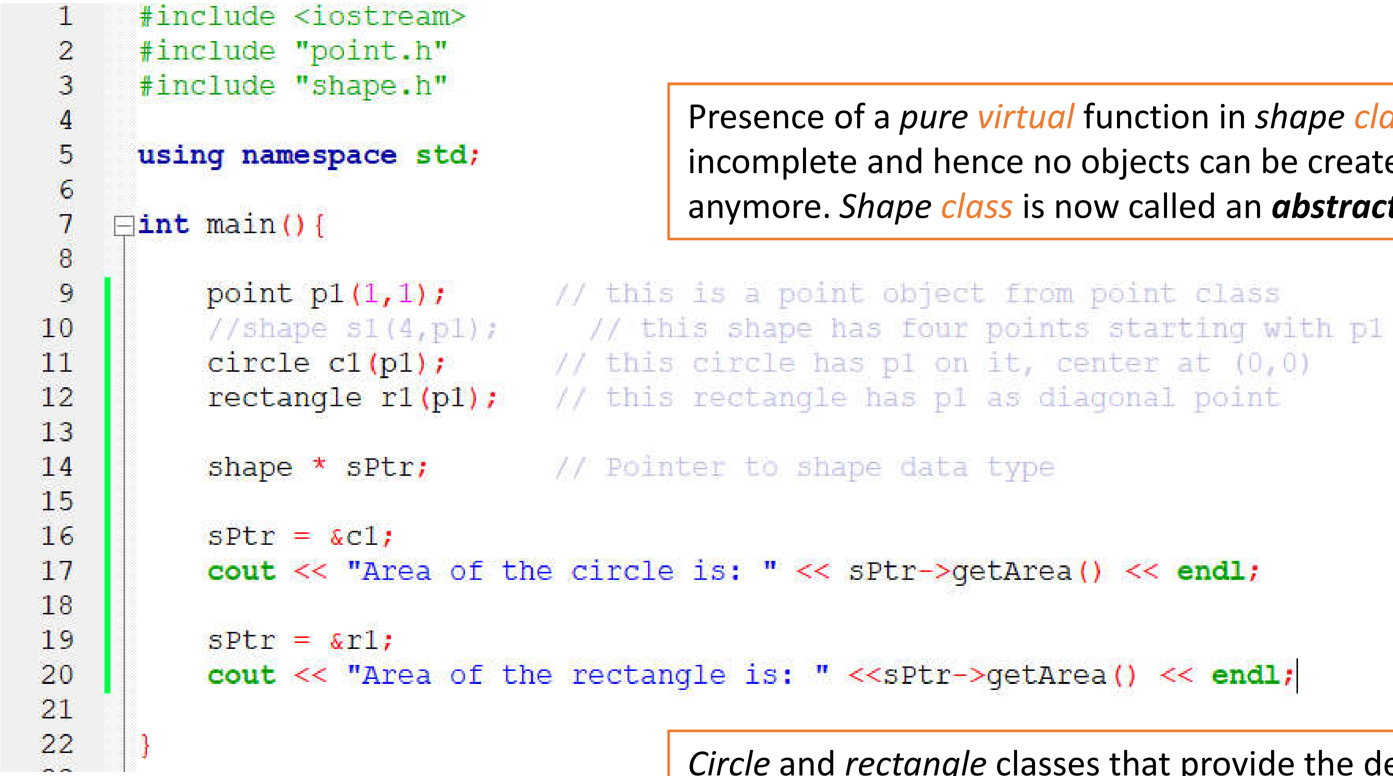
Polymorphism by *virtual* function



```
class circle : public shape{

    double  area;

    public:

        circle();
        circle(const point&);
        ~circle();

        virtual double getArea() const;
        double calcArea();

};

class rectangle : public shape{

    double  area;

    public:

        rectangle();
        rectangle(const point&);
        ~rectangle();

        virtual double getArea() const;
        double calcArea();

};
```

Optionally we can also declare the *getArea()* in both the derived classes as *virtual*.

Polymorphism by *virtual* function

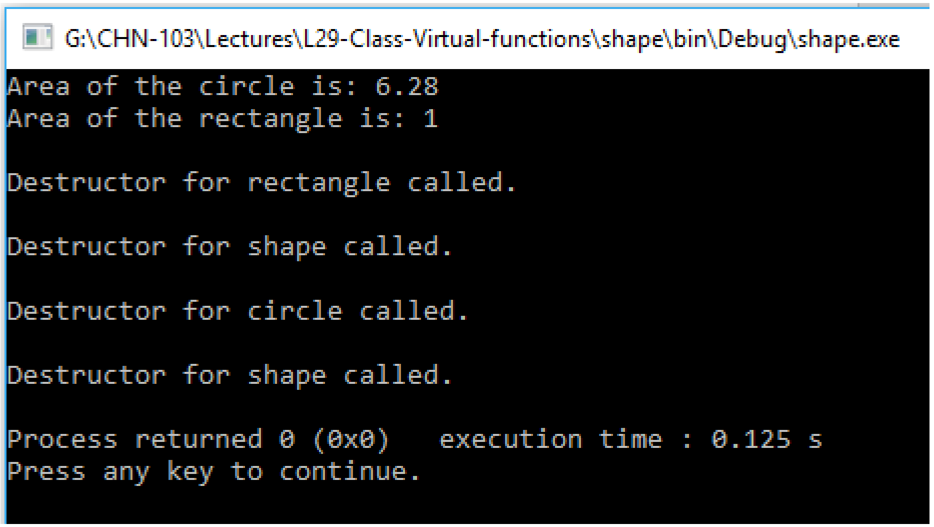


```cpp
#include <iostream>
#include "point.h"
#include "shape.h"

using namespace std;

int main(){

    point p1(1,1);       // this is a point object from point class
    //shape s1(4,p1);     // this shape has four points starting with p1
    circle c1(p1);       // this circle has p1 on it, center at (0,0)
    rectangle r1(p1);    // this rectangle has p1 as diagonal point

    shape * sPtr;        // Pointer to shape data type

    sPtr = &c1;
    cout << "Area of the circle is: " << sPtr->getArea() << endl;

    sPtr = &r1;
    cout << "Area of the rectangle is: " <<sPtr->getArea() << endl;

}
```

Presence of a *pure virtual* function in *shape class* makes it incomplete and hence no objects can be created using it anymore. *Shape class* is now called an ***abstract class***.

Circle and *rectangle* classes that provide the definition of the *virtual* functions are called ***concrete class*** and object can be created from them.

Polymorphism by *virtual* function



G:\CHN-103\Lectures\L29-Class-Virtual-functions\shape\bin\Debug\shape.exe

```
Area of the circle is: 6.28
Area of the rectangle is: 1

Destructor for rectangle called.

Destructor for shape called.

Destructor for circle called.

Destructor for shape called.

Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

Here we are able to get different output from different objects depending on which object is pointed by base class pointer. This is *polymorphic behavior.*

The destructor also are added to show that every derived class object has a base class object created and is destroyed after the derived class object is destroyed.