Final Course Project
ENSF 607 Winter 2020

Polymorphism and Inheritance in C++

Pablo Adames


April 20, 2020

1 Exercise 1: Following inheritance with an AR diagram (20 marks)

1.1 Memory diagram at /* POINT ONE */ (14 marks)

The goal is to draw a memory diagram for the state when the program execution reaches the comment `/* POINT ONE */` seen on Listing 1. The initial analysis tells that at that point the initializer list of the object of type `Sun` has been executed and the member object `plt` has been fully built.

```
1 Planet::Planet(char& c): type(c) { /* POINT ONE */ }
```

Listing 1: Point where memory gets drawn, inside the constructor for the member variable `plt` of type `Planet` where the comment `/* POINT ONE */` appears.

Let's describe how we got to that point in the body of the constructor for `sun`. When the program execution reaches line 4 on main, shown in Listing 2, it starts constructing the object `sun` seen in Listing 3.

```
1 int main()
2 {
3     char c = 'D';
4     Sun sun(39, c);
5
6     sun.print();
7
8     return 0;
9 }
```

Listing 2: Point where memory gets drawn, inside the constructor for `Planet` where the comment `/* POINT ONE */` appears.

```
1     Sun(double x, char& ch):CelestialBody(x, "SKY"), plt(ch){ pi = 3.14;}
```

Listing 3: Constructor declaration for an object of type `Sun`.

The next thing that happens is that the constructor for the base class `CelestialBody` gets called as it is the first item in the initializer list, see constructor declaration in Listing 4.

```
1 CelestialBody::CelestialBody(double x, const char* m ): L(m){
```

Listing 4: Constructor declaration for the class `CelestialBody`, the parent of `Sun`.

After control returns from that constructor the member variable `plt` gets called from the initializer list of `sun`, see Listing 3. Constructing `plt` involves calling the objects in its initializer list, in this case to assign the value `'C'` to the member variable `type`, as seen on Listing 1.

After this, the empty body of the constructor for `plt` of type `Planet` gets executed. There is where the comment `/* POINT ONE */` is found. At this point in the execution of the program, the body of the constructor of the object `sun` has not been executed, therefore that object is not fully built in memory yet.

The final state of objects at this point can be seen in Figure 1.

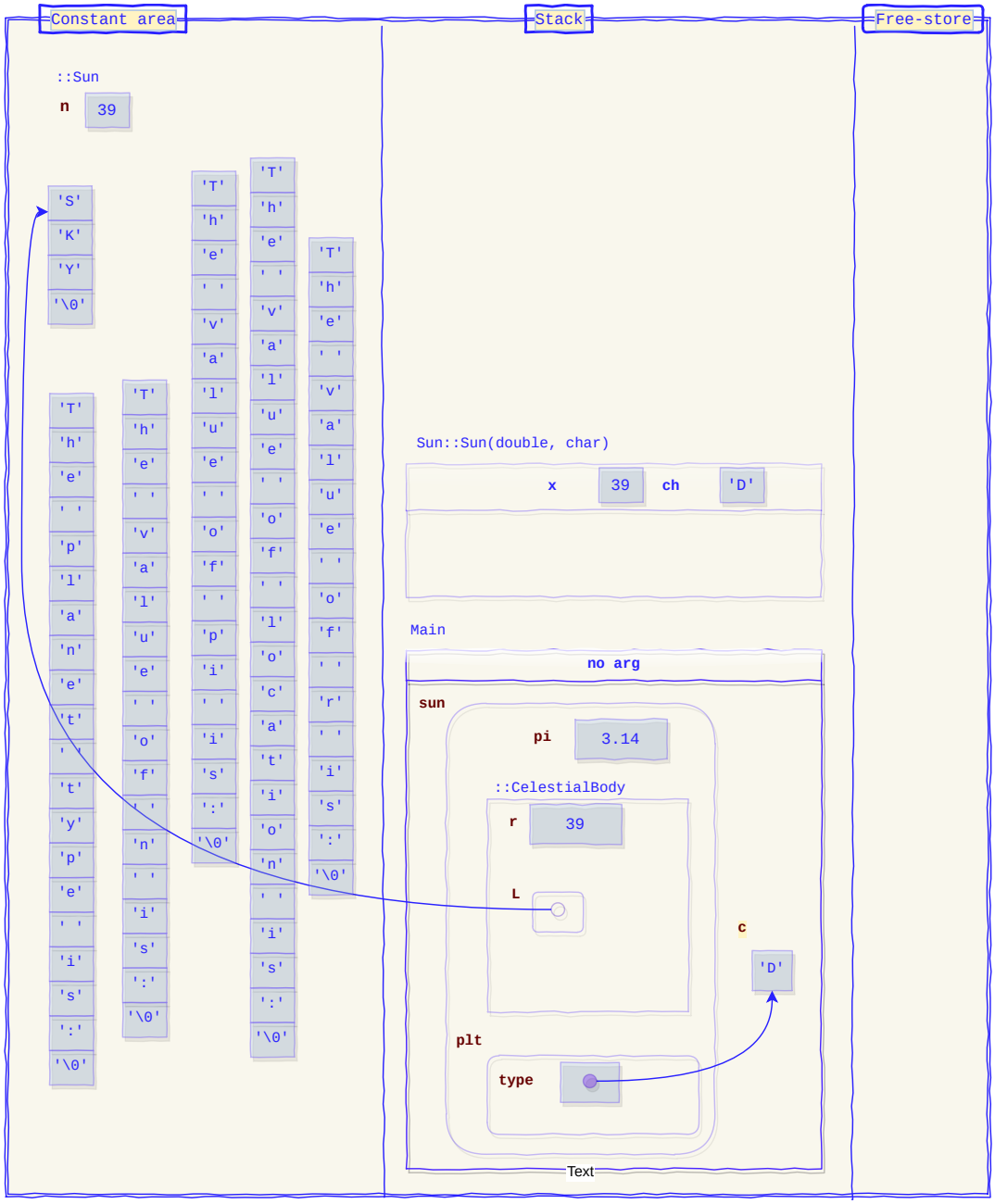

Figure 1: State of objects in memory when the control of flow is inside the constructor for `plt` of type `Planet` where comment `/* POINT ONE */` appears, see Listing 1.

1.2 Memory diagram at /* POINT TWO */ (6 marks)

In order to arrive at the comment where `/* POINT TWO */` in the function `Planet::print()` one starts unwinding the path that starts at line 6 in Listing 2 on page 2, when the object `sun.print()` method is called.

```
1 void Sun::print(){
2     CelestialBody::print();
3     cout << "\nThe value of pi is: " << pi << endl;
4     cout  << "The value of n is: " << n << endl;
5     plt.print();
6 }
```

Listing 5: Declaration of the method `print` for the class `Sun`.

The call to print executes the code seen in listing 5. At line 5 in that code chunk the function `print` of the planet object `plt` gets called. That code is illustrated in Listing 6. The comment `/* POINT TWO */` shows in that function at line 4 of the code chunk.

```
1 void Planet::print(){
2     cout << "The planet type is: " << type;
3
4     // POINT TWO
5 }
```

Listing 6: Declaration of the method `print` for the class `Planet`.

The corresponding memory diagram at that location can be found in Figure 2 on page 5.

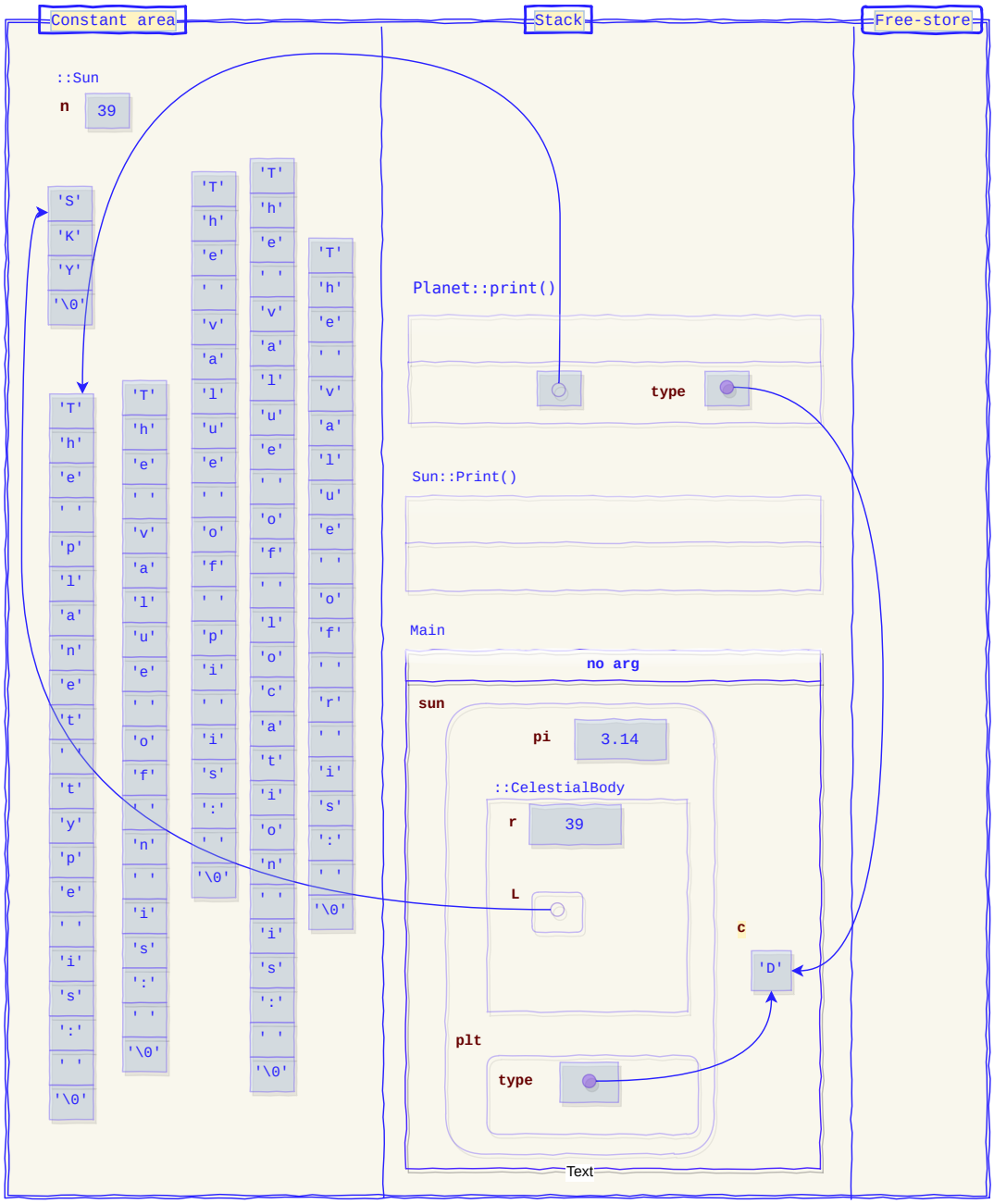

Figure 2: State of objects in memory when the control of flow is in function `Planet::print()` at the comment `/* POINT TWO */`, see Listing 6.

2 Exercise 2

The concepts of aggregation, composition, and inheritance in C++ and java in terms of concepts are similar, but they are completely different in terms of implementation and syntax.

2.1 Part 1: Single Inheritance in C++ (25 marks)

```
using namespace std;
using namespace shape;

void GraphicsWorld::run() {
    cout << endl;
    cout << "This program has been written by: Pablo E Adames" << endl;
    cout << "Submitted at 11:00 pm April 19, 2008" << endl;
    cout << "Testing functions in class Point:" << endl;

    Point m(6,8);
    Point n(6,8);

    n.setX(9);
    m.display();
    n.display();

    cout << endl;
    cout << "The distance between two points as 'm.distance(n)' is " << m.distance(n) << endl;

    cout << endl;
    cout << "Testing the second version of the function distance." << endl;

    cout << endl;
    cout << "The distance between two points as 'distance(m,n)' is " <<  Point::distance(m,n)
    << endl;

    cout << endl;
    cout << "Resting functions in class Square:" << endl;

    cout << endl;
    Square s(5, 7, 12, "SQUARE-S");

    s.display();

    cout << "The area of " << s.getName() << " is: " << s.area() << endl;
    cout << "The perimeter of " << s.getName() << " is: " << s.perimeter() << endl;

    cout << endl;
    cout << "Testing functions of class Rectangle"<< endl;

    cout << endl;
    Rectangle a(5, 7, 12, 15, "RECTANGLE A");
    a.display();
    Rectangle b(16, 7, 8, 9, "RECTANGLE B");
    b.display();
```

Listing 7: Association between and instance of `GraphicsWorld` and the instances of objects of the namespace `shape`.

Association between an instance of the class `GraphicsWorld` and the various objects of

the namespace `shape` can be seen implemented in the Listings 7 and 8. It can be seen as well in the Class Diagram in Figure 8 on page 12.

```
1     cout << "The area of " << a.getName() << " is: " << a.area() << endl;
2     cout << "The perimeter of " << a.getName() << " is: " << a.perimeter() << endl;
3
4     double d = a.distance(b);
5
6     cout << endl;
7     cout << "The distance between two rectangles is: " << d << endl;
8
9     cout << endl;
10    cout << "Testing copy constructor in class Rectangle" << endl;
11
12    Rectangle rec1 = a;
13    rec1.display();
14
15    cout << endl;
16    cout << "Testing assignment operator in class Rectangle" << endl;
17
18    Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");
19    rec2 = a;
20    rec2.display();
21
22    cout << endl;
23    cout << "Testing distance functions between shapes" << endl;
24
25    cout << endl;
26    cout << "The distance between square s' and rectangle rec1' is: " << s.distance(rec1) <<
      endl;
27
28    cout << endl;
29    cout << "The distance between square rec1' and rectangle b' is: " << rec1.distance(b) <<
      endl;
30
31    cout << "The difference between the two distance functions on the same objects is: "
32         << (rec1.distance(b) - Shape::distance(rec1, b) ) << endl;
```

Listing 8: Continuation of the function `GraphicsWorld::run()` showing association between and instance of `GraphicsWorld` and the instances of objects of the namespace `shape`.

The outputs of of Listing 7 can be seen in Figure 3 on page 8 and that of Listing 8 on Figure 4 on page 8.

Single inheritance is used between the classes `Rectangle`, `Square`, and `Shape`. This can be seen in the Class Diagram in Figure 8 on page 12.

```
/home/pablo/Desktop/ENSF607/Project/cpp-poly-project/ex2/cmake-build-debug/ex2

This program has been written by: Pablo E Adames
Submitted at 11:00 pm April 19, 2008
Testing functions in class Point:

X-coordinate: 6
Y-coordinate: 8

X-coordinate: 9
Y-coordinate: 8

The distance between two points as 'm.distance(n)' is 10.8628

Testing the second version of the function distance.

The distance between two points as 'distance(m,n)' is 10.8628

Resting functions in class Square:


Square Name: SQUARE-S

X-coordinate: 5
Y-coordinate: 7
The area of SQUARE-S is: 144
The perimeter of SQUARE-S is: 48

Testing functions of class Rectangle


Rectangle Name: RECTANGLE A

X-coordinate: 5
Y-coordinate: 7

Rectangle Name: RECTANGLE B

X-coordinate: 16
Y-coordinate: 7
```

Figure 3: Console output of the execution of the function `run` on an instance of the class `GraphicsWorld`. Code illustrated in Listing 7.

```
The area of RECTANGLE A is: 180
The perimeter of RECTANGLE A is: 54

The distance between two rectangles is: 11.3578

Testing copy constructor in class Rectangle

Rectangle Name: RECTANGLE A

X-coordinate: 5
Y-coordinate: 7

Testing assignment operator in class Rectangle

Rectangle Name: RECTANGLE A

X-coordinate: 5
Y-coordinate: 7

Testing distance functions between shapes

The distance between square s' and rectangle rec1' is: 8.60233

The distance between square rec1' and rectangle b' is: 11.3578
The difference between the two distance functions on the same objects is: 0
```

Figure 4: Console output of the execution of the function `run` on an instance of the class `GraphicsWorld`. Code illustrated in Listing 8.

2.2 Part 2: Multiple Inheritance (10 marks)

Two new classes are added: `Circle` and `CornerCut`. The class `CornerCut` was used from the funcion `run` of `GraphicsWorld` as shown in Listing 9. The output of the program on the text console appears in Figure 5.

```
1     cout << endl;
2     cout << "Testing functions of CornerCut" << endl;
3
4     CornerCut cc(5.5, 6.2, 10, 20, 5.4, "CORNERCUT CC");
5     cc.display();
6
7     d = cc.distance(c);
8
9     cout << endl;
10    cout << "The distance from the cornercut to the circle is: " << d << endl;
11
12    d = shape::Shape::distance(cc, c);
13
14    cout << endl;
15    cout << "The distance from the cornercut to the circle using the static class function
      distance is: " << d << endl;
16
17    // using array of Shape pointers
```

Listing 9: Use of the class `CornerCut` and the use of poly-morphism through calls to the functions to compute distance between objects of type `Shape`.

```
Testing functions of CornerCut

CornerCut Name: CORNERCUT CC

X-coordinate: 5.5
Y-coordinate: 6.2
Width: 10
Length: 20
Radius of the cut: 5.4

The distance from the cornercut to the circle is: 6.89202

The distance from the cornercut to the circle using the static class function distance is: 6.89202

Process finished with exit code 0
```

Figure 5: Output from using the class `CornerCut`.

The constructor for the class `CornerCut` can be seen in Listing 10 on page 10. In addition to checking if the radius is smaller or equal to the width, an additional check is done for the length to be greater than the width. This check guarantees that the formulas for area and perimeter work accurately. The inheritance from `Circle` to `Shape` as well as from `CornerCut` to `Rectangle`, from `Rectangle` to `Square`, and from `Square` to `Shape` is `public virtual`.

Since there are no default constructors the most specialized class after the multiple inheritance has to build all its ancestors. Thanks to the virtual inheritance mechanism it determines the order of construction of its ancestors and builds a unique base class, in this case a class of type `Shape`.

```
CornerCut::CornerCut(double x, double y, double width, double length, double radius, const
    char *name)
    : Square(x, y, width, name), Rectangle(x, y, width, length, name), Circle(x, y, radius,
    name), Shape(x,y,name) {
    if (radius > width) {
        cerr << "The radius is larger than the width of the CornerCut, aborting!";
        exit(EXIT_FAILURE);
    } else if (length < width) {
        cerr << "The length is smaller than the width of the CornerCut, aborting!";
        exit(EXIT_FAILURE);
    }
}
```

Listing 10: Constructor for the class `CornerCut`. It calls the initialization constructors for all its parent classes. and it validates the input so that the formulas for area and perimeter work well.

```
    Shape * sh[4];
    sh[0] = &s;
    sh[1] = &a;
    sh[2] = &c;
    sh[3] = &cc;

    sh[0]->display();
    cout << endl;
    cout << "The area of " << sh[0]->getName() << " is: " << sh[0]->area() << endl;
    cout << "The perimeter of " << sh[0]->getName() << " is: " << sh[0]->perimeter() << endl;

    cout << endl;
    cout << "The area of " << sh[1]->getName() << " is: " << sh[1]->area() << endl;
    cout << "The perimeter of " << sh[1]->getName() << " is: " << sh[1]->perimeter() << endl;

    cout << endl;
    cout << "The area of " << sh[2]->getName() << " is: " << sh[2]->area() << endl;
    cout << "The perimeter of " << sh[2]->getName() << " is: " << sh[2]->perimeter() << endl;

    cout << endl;
    cout << "The area of " << sh[3]->getName() << " is: " << sh[3]->area() << endl;
    cout << "The perimeter of " << sh[3]->getName() << " is: " << sh[3]->perimeter() << endl;
```

Listing 11: Constructor for the class `CornerCut`. It calls the initialization constructors for all its parent classes. and it validates the input so that the formulas for area and perimeter work well.

There is more code in `GraphicWorld::run()` to test the use of an array of references to objects of type `Shape`. This array is filled with instances of objects of type `Square`, `Rectangle`, `Circle`, and `CornerCut`. Then the array is visited to poly-morphically generate the area and perimeter of each object. The code can be sen on Listing 11 and the output in Figure 6 on page 11.

The last section of the function `GraphicsWorld::run()` uses the copy constructor and assignment operator of objects of class `CornerCut` to test their implementation. The code is shown in Listing 12 on page 11 and the output in Figure 7 on page 11.

Finally, the UML Class Diagram generated from the code can be seen in Figure 8 on page 12. Aggregation of point in Shape and then generalization between children and their parent classes are illustrated in this diagram using UML notation. This diagram is a faithful representation of the code because it was generated from it using the modelling software VisualParadigm.

```
The area of SQUARE-S is: 144
The perimeter of SQUARE-S is: 48

The area of RECTANGLE A is: 180
The perimeter of RECTANGLE A is: 54

The area of CIRCLE C is: 254.469
The perimeter of CIRCLE C is: 56.5487

The area of CORNERCUT CC is: 177.098
The perimeter of CORNERCUT CC is: 49.2
```

Figure 6: Output of the section of the function `run` in the class `GraphicsWorld` showing the use of an array of references to type `Shape` to poly-morphically call the functionality that retrieves the name, the area, and the perimeter from every object.

```
1     cout << endl;
2     cout << "Testing the copy constructor in class CornerCut" << endl;
3     CornerCut rc(cc);
4     rc.display();
5
6     cout << endl;
7     cout << "Testing the assignment operator in class CornerCut" << endl;
8     CornerCut rc2 = cc;
9     rc2.display();
10 }
```

Listing 12: Use of the copy and assignment constructors for the class `CornerCut`.

```
Testing the copy constructor in class CornerCut

CornerCut Name: CORNERCUT CC

X-coordinate: 5.5
Y-coordinate: 6.2
Width: 10
Length: 20
Radius of the cut: 5.4

Testing the assignment operator in class CornerCut

CornerCut Name: CORNERCUT CC

X-coordinate: 5.5
Y-coordinate: 6.2
Width: 10
Length: 20
Radius of the cut: 5.4
```

Figure 7: Output of the section of the function `run` in the class `GraphicsWorld` showing the use of the copy constructor and assignment operator to construct objects of type `CornerCut`.

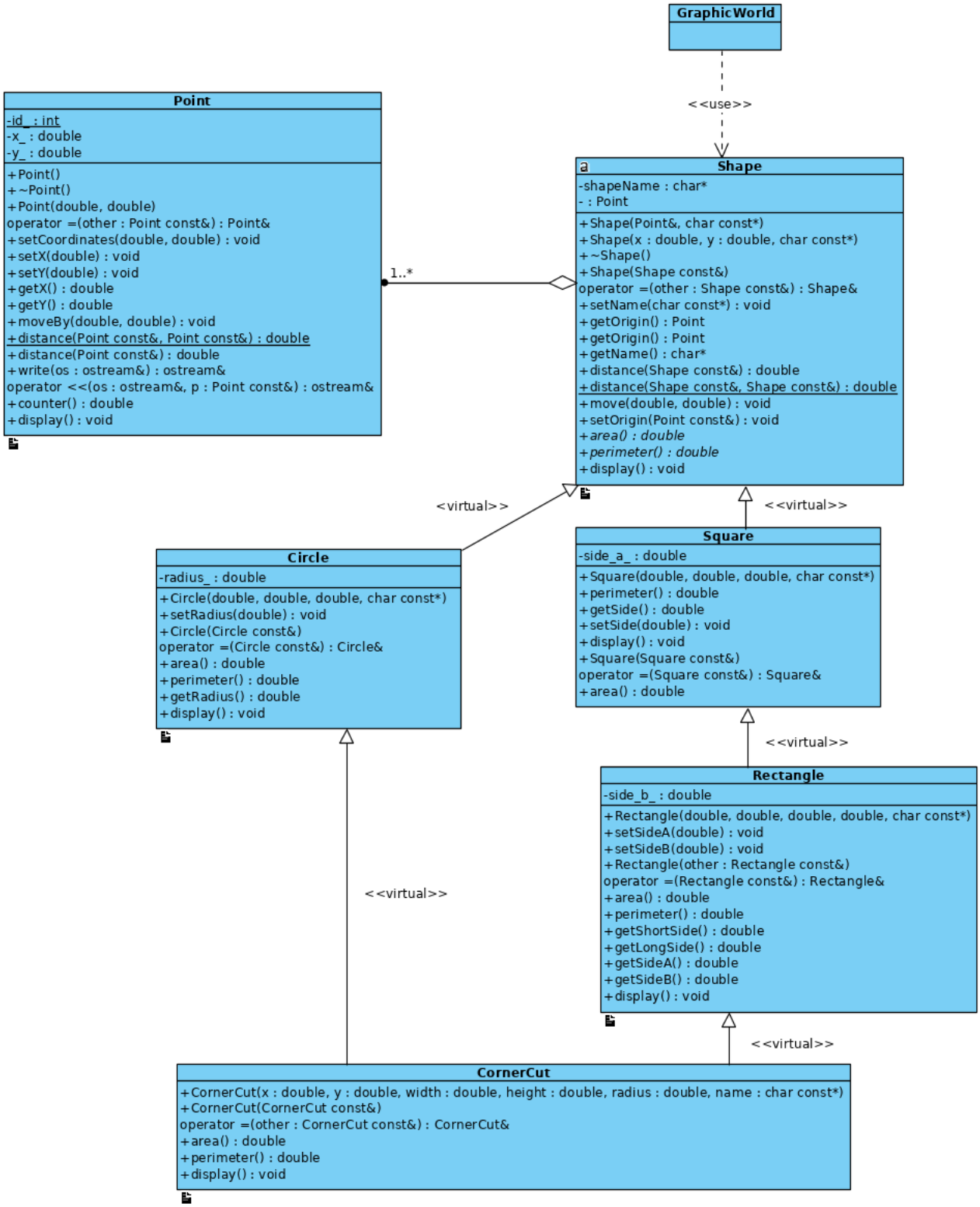

Figure 8: UML class diagram generated from C++ code using *VisualParadigm* software. The class `Shape` is abstract as indicated by the `a` on the top left. The generalization arrows for inheritance are annotated when specified as `virtual`. The static class members are underlined.