LAB11: More on Polymorphism

More Shapes with Polymorphism

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

04/28/2022

Lab11: More Shapes with Polymorphism

- Based on the base class Shape for Lab 10 below, add a new derived class Line. A line is defined by two points, including all the points between these two points. All pure virtual functions in Shape should be inherited.

```
class Shape
{
public:
   Shape() { };
   virtual double area()  const = 0;
   virtual bool inside(const Pt &) const = 0 ;
   virtual double perimeter() const = 0;
   virtual bool degenerate() const = 0;
   virtual void print() const =0;
};
```

- Add a pure virtual function bool intersection(const Shape*) const to the abstract class Shape. This function should return true if the given Shape object intersects the Shape object that makes such a function call. Otherwise, it returns false.

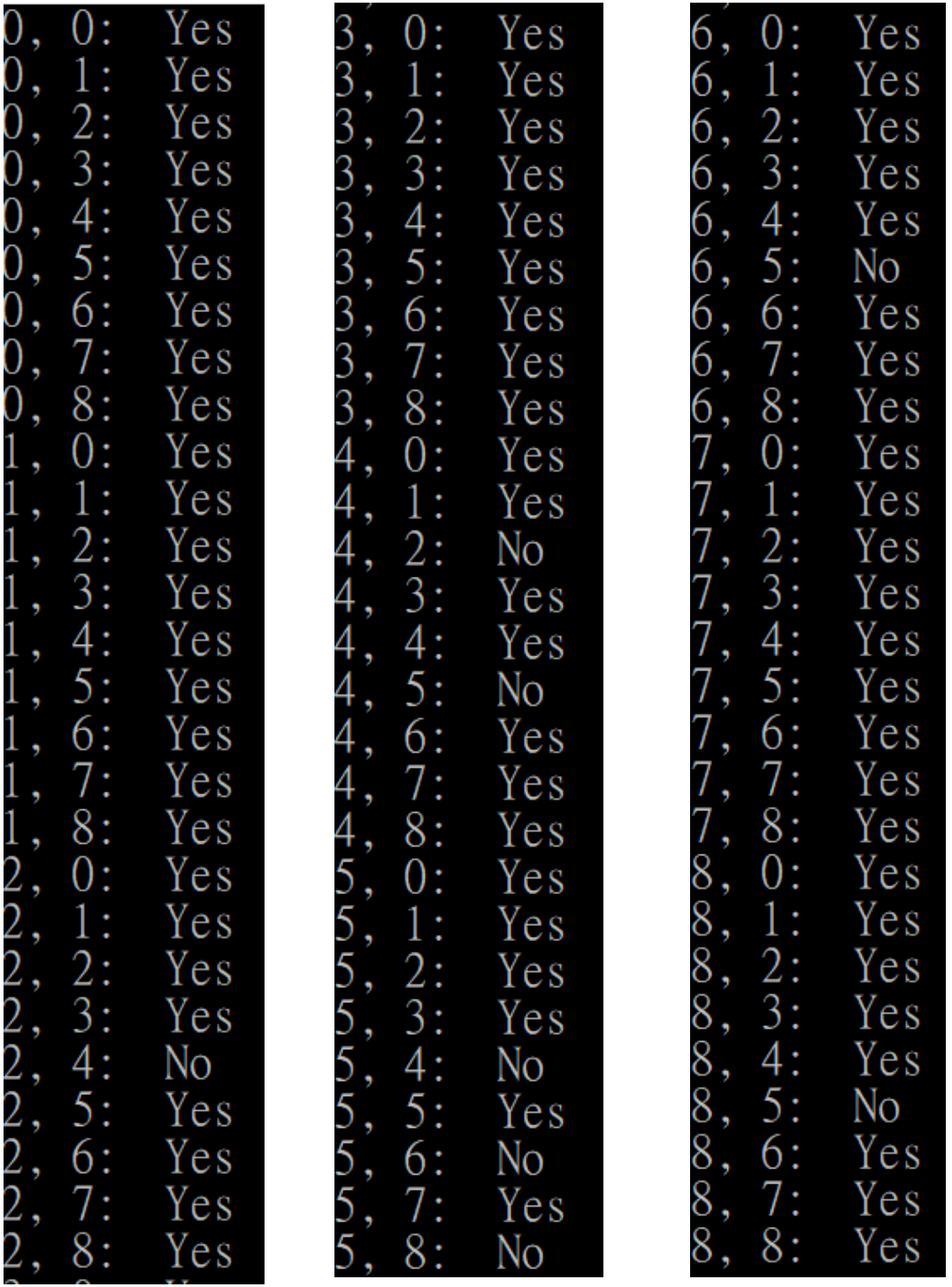
Requirements

- **virtual bool intersection(const Shape*) const = 0 should be added into class Shape.**
- **Main() function should not be modified.**
- **Your output should be exactly the same as given in the example output.**

main() Function

```
int main()
{
    Pt p1(1, 2); // a point defined by X and Y coordinates
    Pt p2(5, 8);
    Pt p3(-2, 10);
    Pt p4(0,0);
    Pt p5(1.5, 4);
    Pt p6(2, 8);
    Pt p7(10, 8);
    Pt p8(0, 8);
    Triangle tr1(p1, p2, p3);  // defined by giving three points
    Triangle tr2(p2, p3, p4);
    Triangle tr3(p2, p6, p7);
    Rectangle rect1(p1, p3); // defined by giving two points
    Rectangle rect2(p1, p1);
    Circle cir1(p4, 10.0);  // defined by a center point and  radius
    Circle cir2(p2, 4);
    Rectangle rect3(p1, p2);
    Triangle tr4(p5, p6, p8);
    const int numShape = 9;
    vector < Shape*> baseShape(numShape);
    baseShape[0] = &tr1;
    baseShape[1] = &tr2;
    baseShape[2] = &rect1;
    baseShape[3] = &cir1;
    baseShape[4] = &tr3;
    baseShape[5] = &rect2;
    baseShape[6] = &cir2;
    baseShape[7] = &rect3;
    baseShape[8] = &tr4;
    for(int i=0; i<numShape; i++)
        for (int j=0; j<numShape; j++){
        if(baseShape[i]->intersection(baseShape[j]))
            cout << i << ", " << j << ":  Yes:" << endl;
        else
            cout << i << ", " << j << ":  No:" << endl;
        }
    return 0;
}
```

Output


0, 0: Yes
0, 1: Yes
0, 2: Yes
0, 3: Yes
0, 4: Yes
0, 5: Yes
0, 6: Yes
0, 7: Yes
0, 8: Yes
1, 0: Yes
1, 1: Yes
1, 2: Yes
1, 3: Yes
1, 4: Yes
1, 5: Yes
1, 6: Yes
1, 7: Yes
1, 8: Yes
2, 0: Yes
2, 1: Yes
2, 2: Yes
2, 3: Yes
2, 4: No
2, 5: Yes
2, 6: Yes
2, 7: Yes
2, 8: Yes
3, 0: Yes
3, 1: Yes
3, 2: Yes
3, 3: Yes
3, 4: Yes
3, 5: Yes
3, 6: Yes
3, 7: Yes
3, 8: Yes
4, 0: Yes
4, 1: Yes
4, 2: No
4, 3: Yes
4, 4: Yes
4, 5: No
4, 6: Yes
4, 7: Yes
4, 8: Yes
5, 0: Yes
5, 1: Yes
5, 2: Yes
5, 3: Yes
5, 4: No
5, 5: Yes
5, 6: No
5, 7: Yes
5, 8: No
6, 0: Yes
6, 1: Yes
6, 2: Yes
6, 3: Yes
6, 4: Yes
6, 5: No
6, 6: Yes
6, 7: Yes
6, 8: Yes
7, 0: Yes
7, 1: Yes
7, 2: Yes
7, 3: Yes
7, 4: Yes
7, 5: Yes
7, 6: Yes
7, 7: Yes
7, 8: Yes
8, 0: Yes
8, 1: Yes
8, 2: Yes
8, 3: Yes
8, 4: Yes
8, 5: No
8, 6: Yes
8, 7: Yes
8, 8: Yes

Key Points for TA Grading

- **The base class Shape should include bool intersection(const Shape &) const.**
- **The derived class Line should be implemented.**
- **The output should be the same as that given in the lab.**
- **main() should not be modified.**