

Section, Name and ID#:

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
CS 120 Introduction to Object-Oriented Programming
QUIZ 09

Date / Time:

Friday, April 14 2017 at 17:00

Duration:

1 hour

Attention:

ANY TYPE OF COMMUNICATION IS STRICTLY PROHIBITED

Write down your section, name and ID# at the top of all used pages

Instructions:

1. Write the solutions in the top half of each page under problem statements
2. Copy the same solution in the bottom section to take with you after quiz
3. Turn your solution into a program, compile and submit the errors
4. Correct the errors and submit the working version of your program

Submission Deadline:

Sunday, April 16 2017, before 22:00

Submission Contact:

skhachat@aua.am

arshavir.voskanyan@gmail.com, nareh_salmasian@edu.aua.am

Problem 1: A rectangle with sides parallel to x and y axes can be represented by its diagonal of type *line*. Implement a C++ class rectangle (its member functions) assuming the existence of all necessary functions of the class *line*:

```
class rectangle
```

```
{ public:
```

```
    rectangle(double x0, double y0, double x1, double y1); // initializes by  
                                                                //bottom-left and top-right coordinates
```

```
    double perimeter();
```

```
    double area();
```

```
    bool intersect(rectangle &that); // checks if the rectangles intersect
```

```
    rectangle union(rectangle &that); // returns least rectangle that includes both
```

```
private:
```

```
    line diagonal; // arrays of x and y coordinates of vertices respectively
```

rectangle::rectangle (double x0, double y0, double x1, double y1):

```
{  
    x0 = diagonal.get-x();  
    y0 = diagonal.get-y();  
    x1 = diagonal.get-x1();  
    y1 = diagonal.get-y1();  
}
```

```
double rectangle::perimeter()
```

```
{  
    return 2 * (y1 - y0) + 2 * (x1 - x0);  
}
```

Student's copy

```
double rectangle::area()  
{  
    return (y1 - y0) * (x1 - x0);  
}  
bool rectangle::intersect(rectangle &that)  
{  
    if (intersect(y0, y1, y0, y1, x0, x1, that.diagonal.get-x(), that.diagonal.get-y1()))  
        return true;  
    return false;  
}
```


Section, Name and ID#:

Problem 2: Implement a C++ *class triangle* (only its member functions marked by **TODO**) the header file of which is given below. The Heron's formula is $area = \sqrt{p(p-a)(p-b)(p-c)}$, where p is the half-perimeter and a, b and c are the sides.

```
class triangle
{
public:
    triangle(double vertex[][3]); // TODO - initializes vertices by specified
    // array of two rows and three columns
    double get_x(int vertex); // returns x coordinate of specified vertex
    double get_y(int vertex); // returns y coordinate of specified vertex
    double side(int vertex); // returns side length from specified vertex to next one

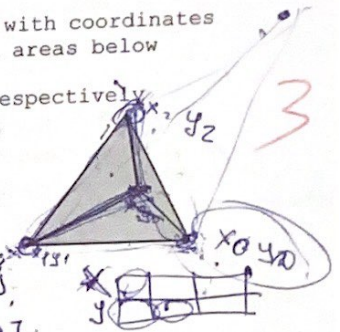
    double perimeter(); // TODO
    double area(); // TODO - computes area using Heron's formula
    bool is_inside(double px, double py); // TODO - checks if a point with coordinates
    // (px, py) is inside the triangle - see shaded areas below

private:
    double x[3], y[3]; // arrays of x and y coordinates of vertices respectively
};
```

triangle::triangle (double vertex[][3]).

```
{
    x[0] = get_x(0);
    x[1] = get_x(1);
    x[2] = get_x(2);
    y[0] = get_y(0);
    y[1] = get_y(1);
    y[2] = get_y(2);
}
```

```
x[0] = vertex[0][0];
x[1] = vertex[0][1];
x[2] = vertex[0][2];
y[0] = vertex[1][0];
y[1] = vertex[1][1];
y[2] = vertex[1][2];
```



double triangle::perimeter() {

return p; ~~double p = (side(0) + side(1) + side(2));~~

double triangle::area() {

return $\sqrt{p/2 \times (p/2 - \text{side}(0)) \times (p/2 - \text{side}(1)) \times (p/2 - \text{side}(2))}$;

Student's copy

~~double triangle::perimeter() {~~
~~return~~

bool is_inside(double px, double py) {

double ~~area1~~ = dist(x0, y0, px, py) + dist(x1, y1, px, py) + dist(x2, y2, px, py) < side(2);

Use the backside, if needed $\text{double area1} = \text{sqrt}(p/2 \times (p/2 - \text{dist}(x0, y0, px, py)) \times p/2 - \text{dist}(x1, y1, px, py)) \times p/2 - \text{side}(2)$;

the same for the other 2 small triangles:

if (area1 < ar) return true; else return false;