EECE.3220: Data Structures

Spring 2017

Programming Assignment #1: Nested Structures

Due **Friday, 2/3/17**, 11:59:59 PM

1. Introduction

This assignment will review the fundamentals of using structures to represent data. It will also familiarize you with the basics of input and output in C++. You will model two-dimensional polygons as a series of points, thus nesting variables of one structure type inside another. Your main program will manage a list of these polygons, as well as giving you a facility to test if a point is inside one of the polygons.

One common algorithm for testing if a point lies inside a polygon is to draw a horizontal line to the right of each point, extending that line to infinity, and count how many times that line intersects polygon edges. If that result is odd, the point is inside the polygon.

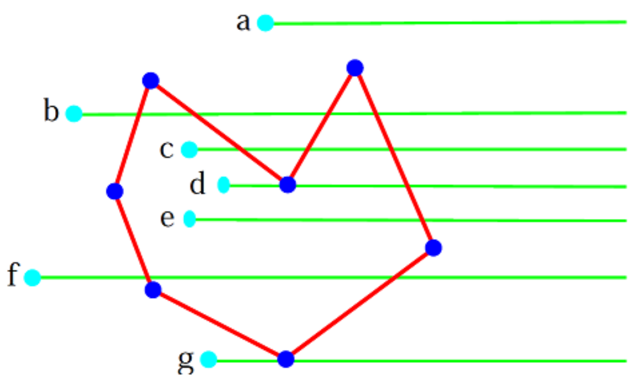

Figure 1: Points both inside and outside a polygon. Points, c, d, and e are inside the polygon--lines extended to the right of each point intersect an odd number of polygon edges.

2. Deliverables

You should submit five files for this assignment:

- ***prog1_main.cpp:*** Source file containing your main function.
- ***Point.h:*** Header file containing definition of Point structure and prototypes of relevant functions.
- ***Point.cpp:*** Source file containing definitions of Point functions.
- ***Polygon.h:*** Header file containing definition of Polygon structure and prototypes of relevant functions.
- ***Polygon.cpp:*** Source file containing definitions of Polygon functions.

Submit all five files by uploading them to your Dropbox folder. Ensure your file names match the names specified above. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

If you have not e-mailed Dr. Geiger for access to a shared Dropbox folder for use in this course, please do so well in advance of the program submission deadline. You will need to register for a Dropbox account on www.dropbox.com if you have not done so already.

3. Specifications

The main program (***prog1_main.cpp***) recognizes four different single-letter commands, most of which call functions described in *Point.h* or *Polygon.h* and defined in the corresponding source file:

- **'A', 'a'**: Add a Polygon to the array of Polygons, which can contain up to 10 elements.
    - When this command is entered, the program should prompt the user to enter the number of vertices in the Polygon (with a maximum of 20 vertices), then read all of those Points and store them in the Polygon structure.
- **'P', 'p'**: Print the entire list of Polygons as a collection of Points. For each Polygon, four coordinates should be shown per line.
- **'T', 't'**: Test whether a Point is inside one of the Polygons.
    - When this command is entered, the program should prompt the user to enter a Point (x-coordinate first), then an index into the Polygon array (
- **'X', 'x'**: Exit the program.

Point.h contains the definition of the Point structure, which consists of two coordinates, x and y. We used this Point structure definition in an in-class example:

```
struct Point {
   double x;     // X coordinate
   double y;     // Y coordinate
};
```

Point.h should also contain prototypes for any functions related to this type, which you must properly define in ***Point.cpp***. You are required to write at least the following two functions (but are welcome to add any others you find necessary):

`void printPoint(Point &pt);`

Print the x and y coordinates of the Point referenced by the argument p using the following format: `(x, y)` (for example, `(3.14, -5.2)`) You do not have to specify a certain number of decimal places, but should ensure that the x- and y-coordinates are enclosed by parentheses and separated by a comma.

`void readPoint(Point &pt);`

Read x and y coordinates from the user input into the Point referenced by the argument p. This function should prompt the user for coordinates.

3. Specifications (continued)

Polygon.h contains the definition of the Polygon structure, which can contain up to 20 Point structures representing the vertices of the Polygon. Note that:

- Your Polygon structure must contain, at a minimum, an array of Point structures and a variable indicating the actual number of vertices in the Polygon.
- Defining a "bounding box" for the Polygon (the minimum and maximum x and y values across all vertices) makes it easier to quickly determine if many points are outside the Polygon—anything outside the bounding box is outside the Polygon.
- Polygon edges are defined to be between consecutive pairs of points stored in the Polygon structure. For example, if the user enters a triangle, that Polygon will contain three Points, with edges between the 1st/2nd Points, 2nd/3rd Points, and 3rd/1st Points.

Polygon.h should also contain prototypes for any functions related to this type, which you must properly define in ***Polygon.cpp***. You are required to write at least the following two functions (but are welcome to add any others you find necessary):

`void printPoly(Polygon &pol);`

Print all of the vertices of the polygon in order, with at most 4 vertices per line. For example, given a polygon with 6 vertices, the output of this function might be:

```
Vertices:
(3, 2), (5.6, 7.8), (4, 3), (-1, -1)
(0.9, 0.8), (0, 2)
```

`void readPoly(Polygon &pol);`

Prompt for and read the number of vertices in the Polygon, then read all of those Points.

4. Hints

Point-in-polygon algorithm: A few pointers on how to test for a point inside a polygon:

- As noted above, you can quickly determine if a point is outside a polygon if it's outside the box determined by the minimum and maximum x- and y-coordinates.
- To implement the algorithm in Section 1, note the following:
    - The "infinite line" to the right of a given test point has the same y-coordinate for every point on that line, with the x-coordinates of the entire line being greater than or equal to the initial point.
    - Testing if this imaginary line crosses a polygon edge therefore requires you to do the following for each edge:
        - Check that the y value for the test point is between the y values of the edge endpoints.
        - Check that the x value for the test point is to the left of the line connecting those edge endpoints.
    - Remember, a polygon edge is defined by consecutive pairs of Point structures inside the Polygon structure.

5. Test Cases

Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your assignments, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

The test cases were generated in Xcode, so I've copied and pasted the output below, rather than showing a screenshot of the output window. User input is underlined, although it won't be when you run the program.

```
Enter command (A | P | T | X): A
Enter number of vertices: 4
Point 0--Enter x and y coordinates: 0 0
Point 1--Enter x and y coordinates: 1 5
Point 2--Enter x and y coordinates: 7 6
Point 3--Enter x and y coordinates: 4 -1

Enter command (A | P | T | X): A
Enter number of vertices: 6
Point 0--Enter x and y coordinates: 3 2
Point 1--Enter x and y coordinates: 5.6 7.8
Point 2--Enter x and y coordinates: 4 3
Point 3--Enter x and y coordinates: -1 -1
Point 4--Enter x and y coordinates: 0.9 0.8
Point 5--Enter x and y coordinates: 0 2

Enter command (A | P | T | X): P
POLYGON 0:
Vertices:
(0, 0), (1, 5), (7, 6), (4, -1)

POLYGON 1:
Vertices:
(3, 2), (5.6, 7.8), (4, 3), (-1, -1)
(0.9, 0.8), (0, 2)

Enter command (A | P | T | X): T
Point to test--Enter x and y coordinates: 2 3
Enter polygon number: 0
(2, 3) is inside polygon 0

Enter command (A | P | T | X): T
Point to test--Enter x and y coordinates: 2 3
Enter polygon number: 1
(2, 3) is inside polygon 1
```

5. Test Cases (continued)

```
Enter command (A | P | T | X): T
Point to test--Enter x and y coordinates: -0.5 -0.5
Enter polygon number: 0
(-0.5, -0.5) is outside polygon 0

Enter command (A | P | T | X): T
Point to test--Enter x and y coordinates: -0.5 -0.5
Enter polygon number: 1
(-0.5, -0.5) is inside polygon 1

Enter command (A | P | T | X): x
```