Data Structures and Objects
CSIS 3700

Spring Semester 2020 — CRN 21212 / 21213

Project 1 — Inside the Polygon
Due date: Friday, February 14, 2020

Goal

Develop and implement a **`Fraction`** class and a **`Point`** class and use them to determine whether or not a given point is inside a given polygon.

Details

▹*The `Fraction` class*

For this project, you should use the **`Fraction`** class you created in lab. Make sure that all methods have been implemented *and tested*.

▹*The `Point` class*

Write a **`Point`** class that implements the following:

- Input and output of a point in the form (x, y) where x and y are two **`Fraction`** objects
- Add and subtract two points using **`operator+`** and **`operator-`**
- Multiply two points, computing the cross product
- Multiply a point by a **`Fraction`**, which multiplies both of the point's coordinates by the given fraction.
- Getter methods that return the x-coordinate and y-coordinate.
- Comparison to see if two **`Point`** objects are the same or not the same; this should use **`operator==`** and **`operator!=`**.

Note that both forms of multiply should use **`operator*`**; for cross product the parameter should be a **`Point`** and for scaling it should be a **`Fraction`**.

▹*Source code layout*

In this project you should use a conventional text editor and the **`g++`** and **`make`** tools. Since I want you to get some additional experience with these tools, you should not use an IDE for this project.

The project is also an exercise in *separate compilation*, where the source code is divided into multiple parts. The project should consist of six files:

- A **`fraction.h`** header file that contains the class definition for the **`Fraction`** class;
- A **`fraction.cc`** file containing the **`Fraction`** class methods (functions);

- A `point.h` header file that contains the class definition for the `Point` class;
- A `point.cc` file containing the `Point` class methods (functions);
- A file containing a `main` function that performs the I/O and the main algorithm.
- A `Makefile` that directs the process of creating the executable file

▹*Arithmetic on points*

Suppose you have two points, $\mathbf{a} = (\mathbf{a}_x, \mathbf{a}_y)$ and $\mathbf{b} = (\mathbf{b}_x, \mathbf{b}_y)$. Their sum and difference are defined by taking the sum or difference of their coordinates individually:

$$\mathbf{a} + \mathbf{b} = (\mathbf{a}_x + \mathbf{b}_x, \mathbf{a}_y + \mathbf{b}_y)$$
$$\mathbf{a} - \mathbf{b} = (\mathbf{a}_x - \mathbf{b}_x, \mathbf{a}_y - \mathbf{b}_y)$$

If $r \in \mathbb{R}$ is a real number, then the product of r and $\mathbf{a}$ just multiplies each coordinate by r:

$$r\mathbf{a} = (r\mathbf{a}_x, r\mathbf{a}_y)$$

Note that each of the preceding three operations yields a point as an answer.

It is also possible to "multiply" two points; this is called a *cross product*. The cross product of $\mathbf{a}$ and $\mathbf{b}$ is defined as follows:

$$\mathbf{a} \times \mathbf{b} = \mathbf{a}_x\mathbf{b}_y - \mathbf{a}_y\mathbf{b}_x$$

Note that the cross product yields a single real number as an answer.

▹*The general algorithm*

You begin with a polygon $Q = \langle \mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{k-1} \rangle$ and a point $\mathbf{p}_1$. The polygon consists of a list of vertices; we'll assume that the vertices are listed traveling counterclockwise around the polygon. The question is whether or not $\mathbf{p}_1$ is inside Q; we'll consider being on the edge of Q as not being inside.

Consider a path that may or may not cross some of the edges of Q; if it does cross an edge, we'll assume that (a) it only does so at one point and (b) that point is not one of the polygon's vertices. Then, every time the path crosses an edge, the path crosses from inside the polygon to outside, or vice versa.

Now, take a point $\mathbf{p}_2$ known to be outside the polygon, and the point in question $\mathbf{p}_1$, and let our path be the line segment $\overline{\mathbf{p}_1\mathbf{p}_2}$. If the segment never crosses an edge of the polygon, then the entire path must lie outside of the polygon, and $\mathbf{p}_1$ is outside. If the segment crosses only one edge then one endpoint is outside the polygon and the other — which must be $\mathbf{p}_1$ — is inside. Continuing with this process, it is easy to see that $\mathbf{p}_1$ is inside the polygon Q if and only if the line segment crosses an odd number of polygon edges.

There are some issues that must be addressed for this approach to work. First, $\mathbf{p}_2$ must be outside of Q. Suppose y_{max} is the largest y-coordinate of any $\mathbf{q}_i \in Q$. Then, any point $(x, y_{max} + 1)$ must be outside of Q.

The second issue is that we must guarantee the two assumptions about edge crossing are always true. To help with this, we enlist the help of the cross product. If $(\mathbf{q}_i - \mathbf{p}_1) \times (\mathbf{p}_2 - \mathbf{p}_1) = 0$, then if $\overline{\mathbf{p}_1\mathbf{p}_2}$ intersects either $\overline{\mathbf{q}_{(i-1) \bmod k}\mathbf{q}_i}$ or $\overline{\mathbf{q}_i\mathbf{q}_{(i+1) \bmod k}}$, then it will intersect both at $\mathbf{q}_i$. This will result in an incorrect number of edges crossed.

To resolve this issue, use the following algorithm.

Algorithm 1 Finding an acceptable point outside polygon Q

```
1:  procedure FindOutsidePoint($Q$, $\mathbf{p}_1$)
2:      $y_{max} \leftarrow$ largest $y$-coordinate of any $\mathbf{q}_i \in Q$
3:      $y_{max} \leftarrow y_{max} + 1$                          ▷ Correction!
4:      $x \leftarrow 0$                              ▷ Outside point $\mathbf{p}_2 = (x, y_{max})$

5:      do
6:          $x \leftarrow x + 1$                                  ▷ Correction!
7:          $good \leftarrow true$
8:          for $i \leftarrow 0$ to $k - 1$ do
9:              if $(\mathbf{q}_i - \mathbf{p}_1) \times (\mathbf{p}_2 - \mathbf{p}_1) = 0$ then
10:                 $good \leftarrow false$
11:             end if
12:         end for
13:     while ($good = false$)

14:     return $(x, y_{max})$
15: end procedure
```

The final issue is checking to see if $\mathbf{p}_1$ is on an edge of the polygon. For the basic project, we will assume that this won't happen. For 10% extra credit for the project, incorporate a check for this situation. It will involve a slight modification of the next algorithm.

▹*Line segments on the plane*

On a Cartesian plane, two line segments may or may not intersect with one another. If they intersect, one or both segments may touch the other at an endpoint, or they may cross somewhere in the middle. If they do not intersect, they could simply be far enough from each other, they could be parallel segments, or they could even be collinear (both segments lie along the same line.)

In this program, we need to know if two line segments intersect or not. This algorithm can provide additional information, such as whether two non-intersecting segments are parallel or not; however, we are only interested in intersecting or not.

Algorithm 2 Determine two line segments intersect

```
procedure INTERSECT($\mathbf{p}_1, \mathbf{p}_2, \mathbf{q}_1, \mathbf{q}_2$)
    $\mathbf{r} = \mathbf{p_2} - \mathbf{p_1}$                ▷ Distance from $\mathbf{p_1}$ to $\mathbf{p_2}$
    $\mathbf{s} = \mathbf{q_2} - \mathbf{q_1}$                ▷ Distance from $\mathbf{q_1}$ to $\mathbf{q_2}$

    $\mathbf{v} = \mathbf{q_1} - \mathbf{p_1}$
    $d = \mathbf{r} \times \mathbf{s}$

    if $d = 0$ then
        return $false$
    else
        $t = \frac{\mathbf{v} \times \mathbf{s}}{d}$          ▷ Intersection point along $P$ in units of $\mathbf{r}$
        $u = \frac{\mathbf{v} \times \mathbf{r}}{d}$          ▷ Intersection point along $Q$ in units of $\mathbf{s}$

        if $0 \le t \le 1$and $0 \le u \le 1$ then
            return $true$
        else
            return $false$
        end if
    end if
end procedure
```

▹*Tying the pieces together*

The program should start by reading the number of vertices in polygon Q; the number of vertices will not exceed `100`. Next, read the vertices and store them in an array. Finally, read the position of point $\mathbf{p}_1$.

Find point $\mathbf{p}_2$, then loop through each edge of the polygon and count the number of edges that intersect line segment $\overline{\mathbf{p}_1\mathbf{p}_2}$. Output an appropriate message about whether or not $\mathbf{p}_1$ is inside or outside of the polygon.

For extra credit, check to see if $\mathbf{p}_1$ is on the border of the polygon and output an appropriate message if it is inside, outside or on the border.

What to turn in

Turn in your source code and `Makefile`.

Examples

▹*Example 1*

Input

```
4
(0/1,0/1)
(0/1,1/1)
(1/1,1/1)
(1/1,0/1)
(1/2,1/3)
```

Output

```
Point is inside the polygon
```

▹*Example 2*

Input

```
4
(0/1,0/1)
(0/1,1/1)
(1/1,1/1)
(1/1,0/1)
(3/2,1/3)
```

Output

```
Point is outside the polygon
```

▹*Example 3*

Input

```
9
(0/1,0/1)
(1/1,0/1)
(1/2,1/2)
(1/1,1/1)
(1/2,3/2)
(1/1,2/1)
(1/2,5/2)
(1/1,3/1)
(0/1,3/1)
(3/4,1/2)
```

Output

```
Point is outside the polygon
```

▹*Example 4*

Input

```
9
(0/1,0/1)
(1/1,0/1)
(1/2,1/2)
(1/1,1/1)
(1/2,3/2)
(1/1,2/1)
(1/2,5/2)
(1/1,3/1)
(0/1,3/1)
(3/4,1/1)
```

Output

```
Point is inside the polygon
```