24-780B—ENGINEERING COMPUTATION

Assigned: Wed. Sept. 14, 2022

Problem Set 3

Due: Tues. Sept. 20, 2022

Shape2D Class

Much of engineering involves managing how geometric entities fit together and interact with each other. In this assignment we will begin to model two-dimensional shapes and create algorithms that allow us to reason about their geometries. Since we are still learning about OpenGL, graphics will play only a small part of this assignment.

Let us model a shape as a sequence of coordinate vertices/points (x_i, y_i) that define a non-self-intersecting closed loop with straight line segments from vertex to vertex. In order to close the loop, we assume that there is an additional line segment from the last vertex to the first vertex. The sequence of points is provided as a simple text file with one coordinate pair on each line. Points can be also be added or removed one at a time.

Shape2D

Create a *Shape2D* class and provide the following (note the function signatures following the commented description):

```
std::vector<Point2D> thePoints; // stores the vertices that define the shape

// default constructor for the class. Initializes member variables only.
Shape2D();

// additional constructor for the class. Takes a pre-created ifstream and uses it to read coordinate
// information for the shape. Note that the method's parameter cannot be a string filename
// because the file may already be in the process of being read. The constructor needs to
// instantiate any constituent objects and/or data structures.
Shape2D(std::ifstream &input);

// adds a coordinate point such that the new point becomes the index-th point . For example,
// an index value of 3 will insert a point between the existing 2nd and 3rd points such that the
// new point becomes the new 3rd point. An index of 1 will insert the new point at the start
// of the shape and an index greater than the number of points will insert the new point as
// the last point. Function returns false only if the new point cannot be added for any reason.
bool addPoint(Point2D newPoint, int index);

// creates a coordinate point with the given coordinates and inserts it into the path such that
// the new point becomes the index-th point. Otherwise, similar to above.
bool addPoint(float newX, float newY, int index);

// creates a coordinate point between vertices index-1 and index and inserts it into the line
// segment at the given ratio (between 0.0 & 1.0). The new point becomes the index-th point.
// Returns false if it cannot insert point (e.g., index is too large or too small, ratio value is
// inappropriate. Ratios close to zero will insert the new vertex near index-1
bool addPoint(int index, float ratio);

// removes the indicated point from the shape. Function returns false only if the point
// cannot be removed for any reason.
bool removePoint(int index);

// draws the shape on the graphics screen using OpenGL.
// Ignore default parameter for now (we'll get to them in PS04)
void paint(bool closed = true, bool filled = false);
```

Shape2D

```
// calculates and returns the length of the perimeter of the shape.
float perimeter();

// used to output all the coordinates of a shape. This can be tricky, so we'll discuss in lecture.
friend std::ostream &operator<<(std::ostream &os, const Shape2D &aShape);
```

Auxiliary Classes (implement these first)

In addition to the *Shape2D* class, we need to create some auxiliary structures and classes. **We will develop much of the code for this functionality during the lectures.**

First, we need a succinct way of modeling a point. We can use a *struct* since there is not much to it.

```
struct Point2D {
   float x;
   float y;
};
```

Line2D

We also need to provide a useful way of reasoning about line segments by creating a Line2D class with several static functions, all public:

```
// returns the distance from one point to the other.
static double getLength(Point2D startPoint, Point2D endPoint);

// returns the distance from one point to the other.
static double getLength(float startX, float startY, float endX, float endY);

// returns true if checkpoint is between startPoint and endpoint.
// Uses a tolerance based on distance from startPoint to endPoint.
static bool isBetween(Point2D startPoint, Point2D endPoint, Point2D checkPoint);

// returns the mathematical intersection point of the two lines
// (does NOT use slopes since vertical lines have undefined slopes)
// return { -INFINITY, -INFINITY } if there is no intersection
static Point2D getIntersection(Point2D lineAstart, Point2D lineAend,
      Point2D lineBstart, Point2D lineBend);
```

Deliverables

4 files (zipped together):

- **Shape2D.h**
- **Shape2D.cpp**
- **Line2D.h**
- **Line2D.cpp**

NOT Required for Submission but Useful for Development

A Shape2DTester.cpp file is provided to you in order to facilitate the testing of your implementation. You should edit/append this file to expand the functioning of your implementation. Feel free to also make use of the sample data files provided.

Pseudo code for each programming challenge is not required for submission, but will likely work its way into your code comments.

Learning Objectives

Use of classes and objects in C++.

Creating and maintaining of sequential data structure (std::vector).

Introductory understanding of graphical data representation and display.

Reading data from a file and understanding how constructors can be effectively used to initialize a data model.

Testing code for accuracy.

Implementing algorithms developed by others.