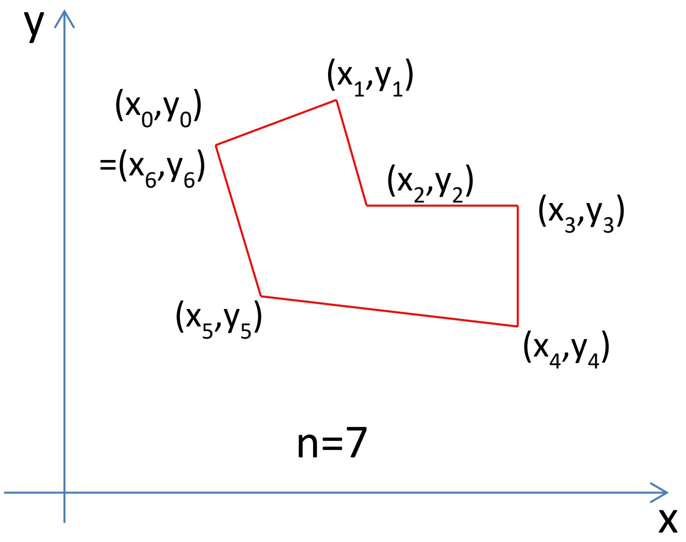
The University of Newcastle
School of Electrical Engineering and Computer Science
SENG2200 Programming Languages & Paradigms
Semester 1, 2018
Programming Assignment 1 – <u>Due Sunday March 25, 23:59pm</u>

Problem Statement:

The area of a polygon, specified on the Cartesian plane by its vertices, is given as follows:

$$A = \tfrac{1}{2} \left| \sum_{i=0}^{n-2} (x_{i+1} + x_i)(y_{i+1} - y_i) \right|$$

e.g.



Note that for a six sided figure, such as the one shown, n is 7, because the last point describing the polygon is equal to the first (it is always the same Cartesian point).

Required Solution:

You are to write a program for a Stock and Station Agent based in Denman, NSW.

Your program will be used to compare different land parcels for their value and how appropriate they are for particular types of agriculture as well as soil conservation and possible pasture improvement. Unfortunately this firm, while willing to trust your programming expertise, finds software development very confronting, and so does not trust things like standard libraries for data structures, and so we haven't even bothered to ask them about things like *generics* being used in data structures or interfaces.

Consequently, all **linked list structures to be used are to be designed and coded from scratch**. You may not use Standard Template Library (STL) or other libraries for data storage manipulation. You may write any auxiliary classes which are necessary to your design and implementation.

Design and write two classes **`Point`** and **`Polygon`**.

The **`Point`** class simply has two double-precision floating point values for x and y coordinate values. It should have a method that will calculate the distance of the point from the origin. Your **`Point`** class should also contain a **`to_string()`** method which will allow the conversion of a **`Point`** object into a String of the form **`(x,y)`** – include the open and close parentheses and the comma in the String. The **x** and **y** values will appear in the string formatted according to the *4.2f* specification; the string returned, will be used for the output of your results.

The **`Polygon`** class contains a sequence of **`Point`** objects representing the ordered vertices of the polygon. Your **`Polygon`** class should contain a **`to_string()`** method which will allow the conversion of a **`Polygon`** object into a String of the form **[point$_0$…. point$_{n-2}$]: area**. For the area, the format to be used is *5.2f* . This will be used for output of your results.

The **`Polygon`** class has a method that will *calculate the area of the polygon*, given by the formula above. You will also need a method to return the *distance from the origin* of the **point$_i$=(x$_i$,y$_i$)** vertex which is closest to the origin. **`Polygon`** will implement the **`ComparePoly`** interface (or your own interface) as per the above, and the **`Polygon`** comparison specification given below.

You are also required to implement a **Circular Doubly-Linked** data structure with the class name of **`MyPolygons`**, using a *single sentinel node* to mark the start/finish of what will become a container for **`Polygon`** objects.

The **`MyPolygons`** class will contain methods to

- **prepend** items into the list (*current* item is the new first in list),
- **append** items into the list (*current* item is the first in list)
- **insert** before a specified (*current*) item,
- step to the **next** item (making it the *current* item),
- **resetting** the *current* item to designate the start of your list, and,
- taking an item from the **head** of the list.

If you think that your container needs more methods than these then we can discuss your proposals on Discussion Board or in class, and update these specifications *if required*.

Input, Required Processing and Output:

The first main task is to read polygon specifications (until end of file/input, from standard input) and place them into an instance of your container, in input order.

Each polygon will be specified in the input by the letter **P**, followed by the number of sides the polygon has (ie n-1 in the formula above), and then pairs of numbers which represent the respective vertices on the Cartesian plane (x-value then y-value), which means vertices $\mathbf{p_0}$ to $\mathbf{p_{n-2}}$ from the above formula. You do not have to worry about any of the data being missing, or out of order. It will be best for your polygon objects to have an array that will contain all n points, that is, explicitly including the last vertex as a copy of the first, as this will allow the easiest implementation of the area formula, but you may use an alternate means of storing the polygon vertices if you wish.

You are then to produce a second list, which contains the same set of **`Polygon`** objects, but this time, sorted into *increasing area order*. This is probably done most simply by placing the polygons into the second list using an insertion sort algorithm.

If any two **`Polygon`** objects have areas *within 0.05%* of each other, then they are assumed to have *equal area*. In these cases of *equal* areas, the polygon with the lower minimum vertex distance from the origin, takes precedence (comes first in your sorted list).

Output for your assignment is a complete print of both your lists, i.e. the polygons in input order, and then the polygons in sorted order, listing the area of each polygon in each case, as per the **`Polygon`** specification of **`to_string()`** given above.

Example input specification for a 6 sided polygon is:

```
P 6 4 0 4 8 7 8 7 3 9 0 7 1
```

That is, the letter P, then the (integer) number of sides (6), then 6 (in this case) pairs of values for the 6 vertices.

i.e. 6 points in total, being points (4, 0) (4, 8) (7, 8) (7, 3) (9, 0) (7, 1)

While this example data only uses *integer* data, *floating point values are possible* for the **x** and **y** values.

Memory Management
You will need to properly conduct memory management.

Written Report:

As you design, write and test your solution, you are to keep track of and report on the following:

1. For each of the programs keep track of how much time you spend designing, coding and correcting errors, and how many errors you need to correct.
2. Keep a log of what proportion of your errors come from design errors and what proportion from coding/implementation errors.
3. Given what we have covered in Topic 3 (*Inheritance*), how could you now treat **Rectangles**, and **Squares** as special cases in this problem?

As a guide, aim for between one and two pages, and format this as a report; not a text file or something.

Submission:

Submission is through the Assessment tab for SENG2200 on Blackboard under the entry **Submit PA1**. If you submit more than once then only the latest will be graded. Every submission must contain a) *an assessment item cover sheet*, b) the *written report*, and c) *your program source files*. Please submit a single file called **`c9999999PA1.zip`** (where 9…9 is your student number); do not submit **`.rar`** or **`.gz`** or any other type of archive file.

Coding Language and Compilation:

Programming is in **C++**, and will be compiled as per the standard lab environment.

Provide a **makefile** that the marker will expect to be able to compile your program with the command **make**, and to run your program with the command **./PA1**, within a **command prompt window**.

Your application will run and take data from a standard text file (in the form given above) which will include multiple polygons definitions – note: that you may assume that that number of points specified for the Polygon will always be correct – and will be specified from the command line, in form **`./PA1 test.dat`**

It is expected that **PA1** will a) be in the *root of your submission folder*, and b) *compile the entire project* without any special requirements (including additional software). You may include a **`readme.txt`** file, if you require any special switches or compilation method to be used; but this may incur a penalty if you stray too far from the above guide lines.

You may only use C++ libraries for input and output. All input to the program will come from *standard input*. All output is to *standard output*.

Note v1.1

All input of polygons will be from **`test.dat`** file. That is, your program should be able to read polygons from the file. The sample format of test.dat file is as follows:

```
P 6 4 0 4 8 7 8 7 3 9 0 7 1
P 3 4 0 4 8 7 8
P 5 4 0 4 8 7 8 7 3 9 0
```

Hint for ComparePoly interface: It is a pure abstract class (so-call “interface” in C++). Make sure you have declared dependent classes before make the interface.

4.2f and 5.2f: They are about the formatting of output. It is in form of **<width>.<precision>**. 5.2f means you reserve **at least** 5 places (including ‘.’) for output and there are two digits for precision. For example, if we have a number 3.456, the output of 5.2f is x3.46, where x is a space.

to_string() methods: You should return a string from the method rather than print it out.