CSE 232

Computer Project #08

Assignment Overview
In this assignment you will practice creating your own user-defined data structures to be used in an ASCII art drawing application. You will utilize what you have learned to date in addition to using classes. It is worth 60 points (6% of the overall grade). It is due 11/12, Monday, before midnight on Mimir. That's two weeks because of the midterm on 11/09(Friday at 7pm. Sorry about that)

Hey, I know you have two weeks but if you wait on this one you will run into the midterm. If you start after the midterm, you will only have a couple of days to finish. Get on this and get it turned in ***early*** and save yourself some time for the midterm.

Background
KTurtle (https://en.wikipedia.org/wiki/KTurtle) is a cool graphics-drawing application that has its own mini-programming-language. For this project, we will be writing a program (in C++), that parses a script written in a version of this mini-language and draws characters on a canvas. To make this simple, we will be drawing ASCII characters on regular output (std::cout). You might call this ASCII art. The programming language is pretty simple, e.g, we won't have colours, different sized cursors, etc. In particular, our mini-scripting-language won't be braced, so you don't have to recognize nested structures. Still, our mini-language will be able to generate some interesting graphics.

A script consists of a set lines where each line is a command. The lines are implicitly numbered (the numbers are not part of the script) starting with Line number 1.

```
1. SetDim 10 12
2. Repeat 100 3-7
3. Draw
4. Move 1
5. Turn 1
6. Move 1
7. Turn 3
```

generates the following:

```
# # # # # #
 # # # # # #
# # # # # #
 # # # # # #
# # # # # #
 # # # # # #
# # # # # #
 # # # # # #
# # # # # #
 # # # # # #
```

Scripting Language
The scripting language has just 7 commands, some of which have arguments. These commands can be used to navigate a pen/cursor around a rectangular 2D toroidal canvas.

By "toroidal" we mean that the edges of the canvas wrap. If you move the pen one up from the top row, it wraps to the bottom row at the same column. If you move the pen one column to the right of the rightmost column, it ends of on the first column of the same row.

Comments: A line can begins with the character '#', in which case that line is a comment and the entire line is ignored and running the program moves on to the next line. The '#' must occur at the beginning of the line.

`1. SetDim X Y`
This command sets the dimension of the canvas to have X rows and Y columns. The default canvas size is 10x10.
Error conditions: This command generates an error if either X or Y is less than 10, or greater than 80. If both the X and Y argument is outside the valid range, only an error for X is generated.

`2. SetPenPosition X Y`
This command sets the position of the pen/cursor to the position (X,Y). The default pen position is (0,0)
Error conditions: This command generates an error if either X or Y is less than 0, or if X is greater than the number of rows of the canvas, or Y is greater than the number of columns of the canvas. If both the X and Y argument is outside the valid ranges, only an error for X is generated.

`3. SetPenSymbol S`
This command sets the pen symbol to a particular character S. The default char is '#'. The symbol is provided as an int that represents the ascii value of the symbol.
Error conditions: The pen symbol must be within the range 32 to 126 (the printable characters)

`4. Draw`
This command draws the current pen symbol at the current pen position. No error conditions.

`5. Move N`
This command moves the pen N positions in the direction the pen is facing. The pen may face in one of 4 directions, up, down, right, or left. The starting direction of the pen is right. No error conditions

`6. Turn N`
This command rotates the pen clockwise 90 degrees N times. e.g. If N is 1 and the current direction is left, then the new direction is up. e.g. if N is 4, the pen faces in the same direction as before. No error conditions

`7. Repeat N B-E`
This command repeats all the commands from line B to line E, a total of N times.
Error conditions:
1. B and E must be valid line numbers in the file.
2. B must come before E.
3. The line number of the repeat command must be less than B. (This avoids scripts that run forever).

Note that these errors must be checked ***in the order listed abvoe***, and only the 1st error that occurs must be logged.

Script Errors
If an error occurs, a message is written to an error log, a std::vector of std::string with an appropriate error message. The state of the canvas at that point unclear and is ignored (we cannot test a canvas that has an error). You will **continue** to run the program and test for any more errors, recording them in the log, though you do not have to update the canvas anymore (after the first error). You will be tested on various error conditions. The test cases will provide you with the proper error message. You will have to follow those error messages exactly to get credit for the Mimir tests.

Specifications
We will work with a class Painter, declared in the provided proj08.h. The declared class requires 4 public functions:

1. `Painter(const &std::string fname)`
A constructor that takes a filename as an argument. This file contains the script that must be parsed. The lines in the file are numbered from 1 onwards.

2. `CreateCanvas()`
This function parses every line from the file, and creates the canvas and the error log. This is where you need to parse the commands, i.e. recognize if the command is valid, and then call the right method to handle that command. This can be done with regular strings, stringstreams (easier than strings), or regexes (even easier, if you are willing to learn regexes). Also, you have to know the line number of each command to be able to report it in the error message.

3. `GetCanvas()`
This function returns the canvas as a std::vector of std::string.

4. `GetErrorLog()`
This function returns the error log as a std::vector of std::string. Note that if this vector is not empty, then it doesn't matter what data is in the canvas. *All* errors must be reported.

These are the only functions that your Painter class *must* provide. Everything else is pretty much up to you. However, the provided proj08.h file contains plenty of suggestions for how to build this class. If you are unsure how to begin, this is the perfect place to start.

Your own .h file
You will be turning in your own proj08.h that has the required elements as well as any other elements you might choose to use. The starter code provides hints, but you have the freedom this time to choose what to do. You need only provide the required elements. For example, you can provide your own private data members, your own private functions. We will only test what we have described above. Feel free to explore.

Testing:
Testing this class is up to you. The provided `main.cpp` is an example of a simple testing framework you could use. Pay attention. Simply running this main file will crash! Read it and you can see how it takes input. The output is written to std::cout, but you can write it to another file very simply, e.g

```
./a.out > file.txt
```

Use the Mimir test cases to figure out the exact spelling of the error messages that must be logged. You are also provided with the tests/ directory that contains the inputs of the tests used in mimir. Write your own tests, to try out different error conditions, or to draw your own patterns!

Hey, no hidden test cases for this one! Yeah!

Deliverables:
You must submit a folder named `proj08/` containing `proj08.cpp` and `proj08.h`. The header (.h) file must contain the declaration of Painter, and the implementation (.cpp) must contain the definitions of the Painter methods.