High Performance Computing

Homework #2

Due: Monday September 18 2017 by 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Sun, Sept 17 2017

Maximum Points: 35

Submission Instructions

This homework assignment must be turned-in electronically via Canvas. Ensure your C++ source code is named `Chart.cpp`, `Chart.h`, `ChartMaker.cpp`, `ChartMaker.h`. You are expected to implement various methods in the two classes (`Chart` and `ChartMaker`) based on the directions in this document. Upload just the four source files you have developed/modified onto Canvas.

Objective

The objective of this homework is to develop a software to manipulate a chart (a list of points) gain experience with:

- Developing idiomatic C++ classes
- Further gain experience with operator overloading
- Working with C++ algorithms library
- Figure out some of the requirements from reference executable

Grading Rubric:

This is an advanced course and consequently the expectations in this course are higher. Accordingly, the program submitted for this homework must pass necessary tests in order to qualify for earning a full score.

NOTE: Program that do not compile, have methods longer than 25 lines, or just some skeleton code will be assigned zero score.

Scoring for this assignment will be determined as follows assuming your program compiles (and is not skeleton code):

- **-1 Point**: for each warning generated by `g++` when compiling your C++ program with the -Wall (report all warnings) compiler flag.
- **-1 Points**: for each warning generated by the CSE departments' C++ style checker (a slightly relaxed version from Google Inc).

NOTE: Points will be deducted for violating stylistic qualities of the program such as: program follows formatting requirements (spacing, indentation, suitable variable names with appropriate upper/lowercase letters, etc). The program includes suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method. Program strives to appropriately reuse as much code as possible.

Starter Code:

In order to streamline this homework the following file(s) are supplied. Do not modify any of these files and do not submit them (your instructor will use the version supplied with this homework for grading purposes):

i. **`Point.h`** `and` **`Point.cpp`**: This is a simple `Point` class except it has been split into a header file (`Point.h`) and a C++ source file (`Point.cpp`). Review the methods in these two files. You will need to follow a similar approach for developing the `Chart` and `ChartMaker` classes for this homework.

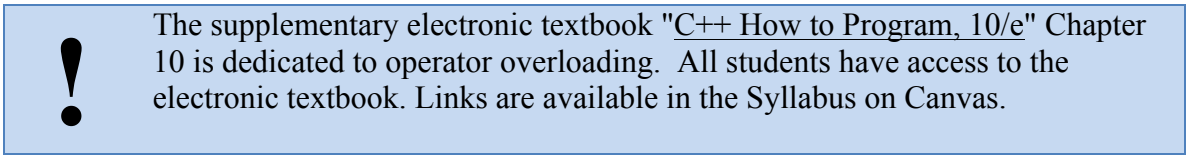
ii. **`main.cpp`**: This is a very simple top-level `main` method that creates a `ChartMaker` object (using the class that you are expected to implement) and calls the `run()` method.

iii. `points.txt`, `small_points1.txt`, `small_points2.txt`: These are simple point data files that can be used for testing. Your instructor will be generating a custom points data file with 1 million points for grading. When working with large sets of points (such as the `points.txt` file) it is best to use `gnuplot` to plot the points to observe if the changes appear correct by suitably adapting (**do not copy-paste!**) the following command-line:

```
$ module load gnuplot
$ gnuplot -e "plot 'points.txt' with linespoints,
'points_copy.txt' with linespoints; pause -1"
```

iv. `ChartMaker`: The final executable that demonstrates the functionality for this homework. You may copy this binary file to Red Hawk and run it on Red Hawk to observe the excepted final behavior of various operations. Note that once you copy the file to Red Hawk you will need to enable executable permissions on this file to run it using the following shell command:

```
$ chmod +x ChartMaker
```



Homework Exercise

This homework exercise involves developing a comprehensive `Chart` class that provides a convenient interface to manipulate a set of points. Each point in a `Chart` is an instance of the supplied `Point` class. You are expected to develop a `Chart` class with the methods documented below. Remember that `Chart.h` contains just the class definition (similar to `Point.h`) and `Chart.cpp` should contain the implementation for the various methods.

API Requirement for Chart class:

Private Instance variable(s) in `Chart`:	
`std::vector<Point> pointList;`	The list of points that constitute this `Chart`. This class has no other instance variables.
Public methods in `Chart`:	
Default (no arguments) constructor	This constructor creates a Chart without any data points in it. Tip: This is a very simple constructor (no statements in its body).
Copy constructor	Copy constructor to make a copy of a given `Chart`. Tip: This is a very simple method-- directly initialize `pointList` from source chart and the body of this method can be left empty.
`Chart(const std::string& fileName);`	Constructor to create a chart using points in a given text file. This constructor creates a chart using points in a text file. The text file is assumed to contain pairs of integers representing the x & y coordinate of each point. An example text files containing data points are supplied along with this homework. The parameter `fileName` indicates the path to the file from where the point data is to be loaded. If this file name is not valid the behavior of this method is undefined (most likely it will throw an exception). Tip: Use `std::copy` to make this method about 3-4 lines long.
`std::ostream& operator<<(std::ostream& os, const Chart& c);`	Stream insertion operator. It should print each point in the chart `c` to the given stream `os` followed by a newline. That is

	each point's x & y coordinates must appear on separate lines reflecting the data organization in the supplied points data file(s) (such as: `points_small1.txt`). Note: For full points body of method (does not include lines for braces) must be no more than 4 lines of properly formatted code.
`std::istream& operator>>(std::istream& is, Chart& c);`	Stream extraction operator. It should load all points (until EOF) from the given input stream into the given chart `c`. The data in the input stream will follow the same organization as the supplied data file(s) (such as: `points_small1.txt`). Note: For full points, body of method (does not include lines for braces) must be no more than 4 lines of properly formatted code.

Assignment operator (`Chart& operator=(const Chart& src);`)	The assignment operator. This method implements the assignment operator to copy the points from the source into `this`. This operator enables assignment of one chart object to another as shown in the example below:
	
	`Chart c1, c2;`
	`// Do some manipulation with c1 and c2`
	`c1 = c2; // c1 & c2 now have same points.`
	
	The parameter `src` is the source chart from where the data points are to be copied. This method returns a reference to `this`.
	
	Tip: This is a 2-line method.

`bool contains(const Point& p) const;`	Method to detect if a given point is in this chart. This method checks to see if a given point `p` is in the list of points associated with this chart (that is the x & y coordinate of `p` is exactly the same as the x & y coordinate of some point in `pointList`). Note: This is a one-liner using the `std::find()` algorithm which is expected for full points.

	This method returns `true` if the given point is in the list of points contained in `this` chart.
`Chart operator+(const Chart& other) const;`	Adds points of two charts together. This method is used to add the points from `other` chart that are not present in `this` chart to create a new chart. Neither `this` nor `other` is modified. For example if `this` chart has points {(1, 10), (2, 20), (3, 30)} and the other chart has points {(2, 10), (2, 30), (3, 30)} the returned `Chart` has points ({1, 10), (2, 20), (3, 30), (2, 10), (2, 30)}. This method must return a new chart object that contains all points from `this` and points from `other` chart. Note: Using the `std::copy_if()` algorithm enables writing this method in 3 lines which is exected for full points.
`Chart operator-(const Chart& other) const;`	Returns a new `Chart` that contains all points in `other` removed from points in `this` chart. This method is used to remove points from `this Chart` that are present in `other Chart` to create a new `Chart` object. Neither `this` nor `other` is modified. For example if `this` chart has points {(1, 10), (3, 30), (2, 20), (3, 30)} and the other chart has points {(2, 10), (3,3 0)} the returned chart has points ({1, 10), (2, 20)}. This method returns a new chart object that contains all points from `this` but not the points in `other`. Note: Using the `std::copy_if()` algorithm enables writing this method in 3 lines (expected for full points)

`Chart operator*(const Point& scale) const;`	Creates a new Chart with x & y coordinates of each point in `this` chart multiplied by the x and y values in `scale`. This method is used to multiply points in `this` chart with the x & y values in the supplied `scale`. Neither `this` nor `scale` object is modified. For example if `this` chart has points {(1, 10), (3, 30), (2, 20)} and the scale is (2, 3) the returned chart has points ({2, 30), (6, 90), (4, 60)}. This method returns a new chart object that contains all points from `this` scaled by the given `scale` factor. Note: Using the `std::transform()` algorithm enables implementing this method with 3 lines of C++ code (expected for full points).
`Chart operator/(const Point& scale) const;`	Creates a new Chart with x & y coordinates of each point in `this` chart is divided by the x and y values in `scale`. This method is used to divide points in `this` chart with the x & y values in the supplied `scale`. Neither `this` nor `scale` object is modified. For example if \c this chart has points {(2, 10), (4, 40), (6, 60)} and the scale is (2, 5) the returned chart has points ({1, 2), (2, 8), (3, 12)}. This method returns a new chart object that contains all points from `this` scaled (or divided) by the given `scale` factor. Note: Using the `std::transform()` algorithm enables implementing this method with 3 lines of C++ code. (expected for full points).

`Chart operator>>(int value) const;`	This member method (not to be confused with stream extraction operator) shifts the points in the chart to the right by the given amount. This method shifts the points in this chart to the right by `value` locations by inserting zeros at the beginning. For example if this chart has points {(1, 10), (2, 30), (3, 30)} then shifting it to the right by 3 results in the new chart having points {(0, 0), (0, 0), (0, 0), (1, 10), (2, 30), (3, 30)} The `parameter` value indicates the number of positions by which the point list is to be shifted to the right. A new Chart object containing points from `this` shifted to the right. Note: This method can be written in four statements using methods in `std::vector`. (expected for full points).
`Chart operator<<(int value) const;`	This member method (not to be confused with stream insertion operator) shifts the points in the chart to the left by the given amount. This method shifts the points in this chart to the left by removing points at the beginning of the list. For example if this chart has points {(1, 10), (2, 30), (3, 30)} then shifting it to the left by `2` results in the new chart having points {(3, 30)}. The parameter `value` indicates the number of positions by which the point list is to be shifted to the right. This method returns a new `Chart` object containing points from `this` chart shifted to the left. Note: This method can be written in three statements using methods in

	`std::vector`. (expected for full points).

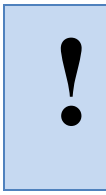
`void analyze(std::ostream& os, const int scale) const;`	Method to display a histogram of distribution of unique points in the 4 quadrants.
	
	This method must generate a histogram illustrating the distribution of all unique points in this chart in 4 different quadrants. For example, if `this` chart contains the points {(1, 1), (-1, 1), (-3, -5), (3, -5), (3, 5), (1, 1), (-3, -5)} then the unique points in the list are {(1, 1), (-1, 1), (-3, -5), (3, 5), (3, -5)} and the number of points in the four quadrants, namely quadrants I, II, III, and IV are: {2, 1, 1, 1} and this method should write the following scaled histogram on the given stream `os`:
	
	`I  : **`
	`II : *`
	`III: *`
	`IV : *`
	
	where the quadrant with the highest frequency is displayed using `scale` number of asterisks and other frequencies are suitably scaled.
	
	Using a different example, assuming that the frequency of occurrence of points in the four quadrants is {500, 750, 250, 400} and the `scale` is 25, then the highest value of 750 is displayed with 25 '*'s while 500 is displayed with (500 * 25 / 750) '*'s, 400 is displayed with (400 * 25 / 750) '*'s and so on.
	
	The parameter `os` is the output stream to which the histogram illustrating the distribution of all unique points in `this` chart is to be printed.
	
	The `scale` parameter indicates the scale factor indicating the number of '*'s to be displayed for the highest frequency of occurrence.

	Note: this method could be implemented using only algorithms without any loops. However, such an implementation is not necessary. Tip: Since there are only 4 quadrants, you can manually repeat statements in the worst-case scenario. The couple of helper classes from functional header that will come-in handy are: `std::bind2nd` and `std::mem_fun_ref`. Refer to online documentation on these two helpers for examples on their usage.

Requirements for `ChartMaker` class:

Once you have developed the `Chart` class the next step is to develop the `ChartMaker` class that provides the user with a convenient menu to read, manipulate, and save charts. The `ChartMaker` essentially uses various methods in `Chart` to perform the necessary manipulations. The `ChartMaker` is an open ended class and you are expected suitably design it such that it works correctly with the supplied `main` function in `main.cpp` and meets the following functional requirements:

1. The `ChartMaker` performs all operations with a chart which will be referred-to as `current` in this document. Possibly this `current Chart` object may be the only instance variable this class may need. All operations performed by this `ChartMaker` will operate on the `current` chart.
2. Operations such as print, save, and load directly use the `current` chart for performing their operations after obtain any necessary inputs (see sample outputs for details) from the user.
3. Operations such as adding to the `current` charts or subtracting from `current` chart must prompt and obtain another file of points for use in these binary operations as shown in the sample output. These operations will result in changes to `current`.
4. Operations such as scaling and shifting must prompt and obtain the scale factors or shift position values from the user (as shown in the sample output) and suitably modify the `current` chart.



The supplied reference executable CharMaker implements all of the features correctly. **Any time you have questions or need clarifications**, you should run the reference executable and determine expected behavior, mimicking typical reverse engineering efforts you will be expected to be able to do in a real job in the industry.

Note that the `ChartMaker` class should be relatively straightforward to implement. You may add as many methods as you see fit. You are not required to use any algorithms for this class and you may use traditional looping constructs. The points will be assigned depending on design of this class and the quality of the code.

Sample Outputs

Expected outputs from multiple independent runs of the completed program are shown below. User inputs are shown in red for improved readability

```
$ ./ChartMaker
Enter command (h for help): h
All commands are single characters.
Valid commands are:
l: Load data from file as current set of points
+: Add data from another to to current points
-: Subtract data from another file from current poitns
*: Scale current points by given pair of values
/: Scale current points by given pair of values
?: Print histogram current point distribution
<: Shift points to left by a given value.
>: Shift points to right by a given value.
s: Save the current set of points to a given file
p: Print current set of points on screen.
h: Print this message
q: Quit
Enter command (h for help): l
Enter path to file: small_points1.txt
Enter command (h for help): p
1 1
-1 1
-1 -1
1 -1
1 1

Enter command (h for help): q
```

```
$ ./ChartMaker
Enter command (h for help): l
Enter path to file: small_points1.txt
Enter command (h for help): +
Enter path to file: small_points2.txt
Enter command (h for help): p
1 1
-1 1
-1 -1
1 -1
1 1
2 2
-2 2
-2 -2
2 -2

Enter command (h for help): q
```

```
$ ./ChartMaker
Enter command (h for help): l
Enter path to file: small_points1.txt
Enter command (h for help): -
Enter path to file: small_points2.txt
Enter command (h for help): p
-1 -1
1 -1

Enter command (h for help): *
Enter values to scale x & y coordinates: 100 200
Enter command (h for help): p
-100 -200
100 -200

Enter command (h for help): /
Enter values to scale x & y coordinates: 25 50
Enter command (h for help): p
-4 -4
4 -4

Enter command (h for help): >
Enter positions to shift: 5
Enter command (h for help): p
0 0
0 0
0 0
0 0
0 0
-4 -4
4 -4

Enter command (h for help): <
Enter positions to shift: 6
Enter command (h for help): p
4 -4

Enter command (h for help): q
```

Testing for grading

Your instructor will be using automated test harness to check if outputs from your program match the expected reference outputs. Tests (conducted at the shell prompt in a terminal) based on the above sample outputs are shown below. The test files are supplied as part of the homework for your convenience and testing.

```
$ ./homework2 < test_inputs1.txt > my_outputs1.txt
$ diff my_output3.txt ref_outputs1.txt
```

The above `diff` command should not generate any output indicating that your output in `my_outputs1.txt` is exactly the same as the expected reference output `ref_outputs1.txt`

```
$ ./homework2 < test_inputs2.txt > my_outputs2.txt
$ diff my_output2.txt ref_outputs2.txt
```

The above `diff` command should not generate any output indicating that your output in `my_outputs2.txt` is exactly the same as the expected reference output `ref_outputs2.txt`

```
$ ./homework2 < test_inputs3.txt > my_outputs3.txt
$ diff my_output3.txt ref_outputs.txt
```

The above `diff` command should not generate any output indicating that your output in `my_outputs3.txt` is exactly the same as the expected reference output `ref_outputs3.txt`

Turn-in:

Submit just the four C++ source files (`Chart.h`, `Chart.cpp`, `ChartMaker.h`, `ChartMaker.cpp`) that meet the requirements of this homework via Canvas. No credit will be given for submitting code that does not compile or just skeleton code. Verify that your program meets all the requirements as stated in the grading rubric. Ensure your C++ source files are named with the stipulated naming convention. Upload all the necessary C++ source files to onto Canvas. Do not submit zip/7zip/tar/gzip files. Upload each source file independently.