CSci 1113: Introduction to C/C++ Programming for Scientists and Engineers Homework 5

Due Date: Tuesday, Mar. 26 before 5:30pm.

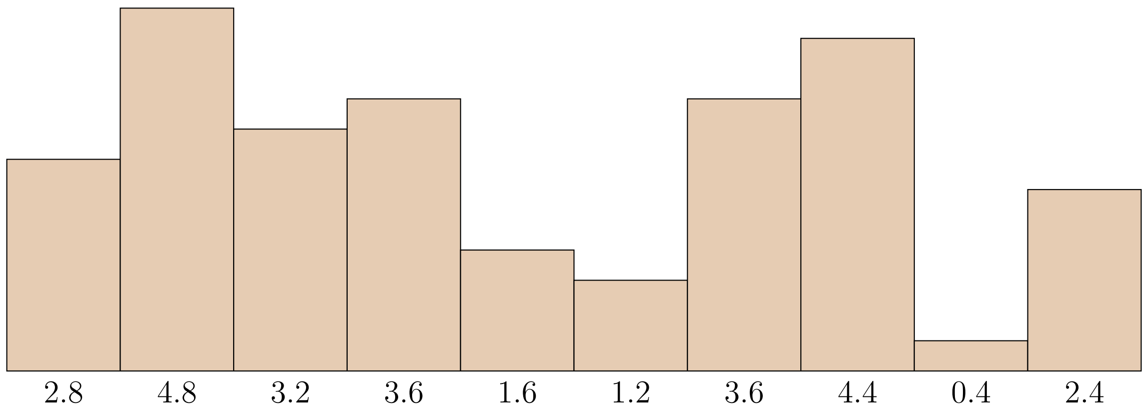
Purpose: This homework involves further practice with arrays.

Instructions: This is an *individual* homework assignment. There are *two* problems worth 30 points each. The rules and reminders are the same as for past homework, so see the previous homework write-ups if you need a reminder.

Problem A: Measuring water storage (30 points)
As geographical measurement data becomes increasingly available, it has become important to have effective algorithms to understand this data. This gives birth to many interesting data analysis problems. One example, is the water storage problem: given a description of the ground heights in a region, compute how much water it could store.

Solving this problem can be surprisingly complicated in the general case, therefore we will consider a simplified version of this problem — the 1 dimensional water storage problem: Given a series of height measurements describing the cross-section of a 2-d shape, how much water (what area) could that cross-section store? In particular, imagine heavy rain falling over the shape — how much water could the given shape hold before all excess water simply flows over the sides? To solve this problem we will be representing the cross-section with a length 10 array of doubles. Each position in the double array represents the height of the shape at one position. With 10 height values we can represent a shape as shown below:

Array Values: {`2.8, 4.8, 3.2, 3.6, 1.6, 1.2, 3.6, 4.4, 0.4, 2.4`}
Cross-section:



Computing the water storage capacity involves first computing the maximum possible water level at any given position. Water will only stay stored in place if there is a wall to both the left and the right. Therefore at any position in the shape we can compute the water level by looking for the highest point to the left and right of that point. The maximum possible water level is then the lesser of the maximum heights to the left and the right. In some cases, the water level will be below the current height, such as when the shape is very tall in one place, or at either end of the shape, where water can simply flow off the edge (meaning the water level would be 0.0). Once we have the water level, the stored water is simply the difference between the water level and the current height.

Program Description Your program should have 4 functions (including main)

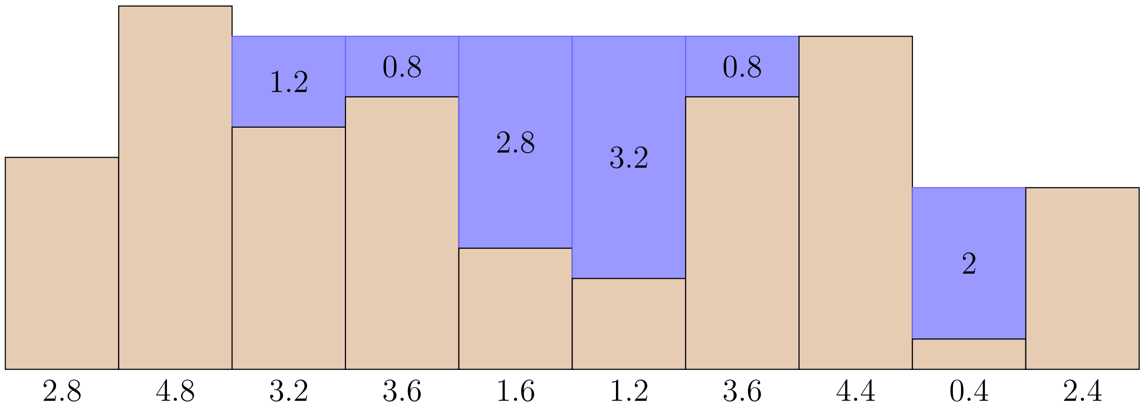
- `leftHeight`. This function should take the heights array and an index as parameters and compute the maximum height to the left of the index. So if the input is {1.0,3.0,2.0,5.0,4.0,3.0,2.0,1.0,6.0,3.0} and index 7 (value 1.0) the largest value to the left would be 5.0. If the index 0 is given, then there are no heights to the left, and `leftHeight` should return 0.0.
- `rightHeight`. This function should take the heights array and an index as parameters and compute the maximum height to the right of the index. So if the input is {1.0,3.0,2.0,5.0,4.0,3.0,2.0,1.0,6.0,3.0} and index 4 (value 4.0) the largest value to the right would be 6.0. If the index 9 (the last valid index in a length 10 array) is given, then there are no heights to the right, and `rightHeight` should return 0.0.
- `waterHeight`. This function should compute the water level at a given index. The water level at that index is the minimum of the rightHeight and leftHeight at that index. This may be lower than the actual height at that index (in which case this location cannot hold any water).
- `main`. You main program should:
  - Ask the user for exactly 10 height values measured as doubles and store it in a length 10 double array.
  - Validate the height values. For this program we will require all height values to be strictly greater than 0.0. Therefore if one or more of the height values are equal or less than 0.0, the program should print "Invalid heights!" and halt.
  - The program should then loop over every position and compute how much water can be stored at that position.
  - Finally, the program should compute and print the total water storage for the shape.

Examples: (Note, user entered text is underlined, all spaces shown are single-spaces. After each example is a drawing of the scenario to help you understand the situation. You are not required to create drawings like this; they are included to help you visualize the problem.

Example 1

```
Input 10 heights:  2.8 4.8 3.2 3.6 1.6 1.2 3.6 4.4 0.4 2.4
Amount of water:  10.8
```

Example diagram:

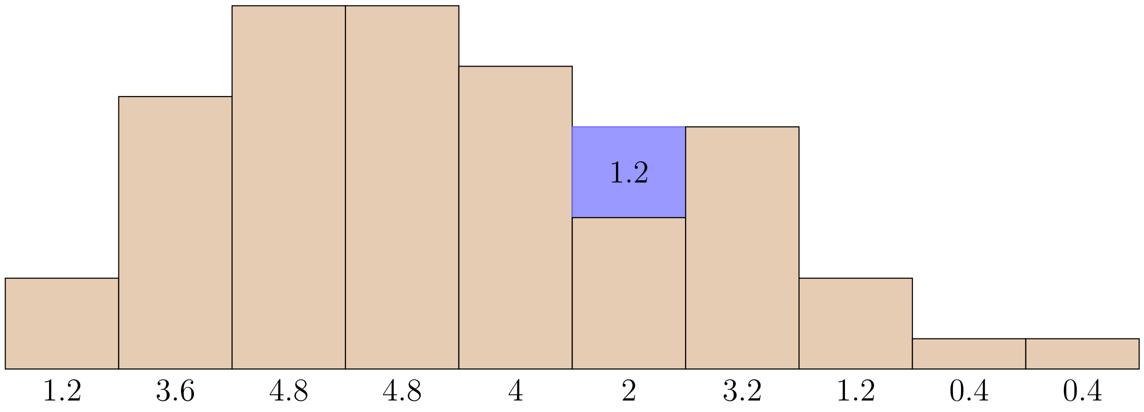


Example 2

`Input 10 heights:` `1.2 3.6 4.8 4.8 4 2 3.2 1.2 0.4 0.4`

`Amount of water: 1.2`

Example diagram:

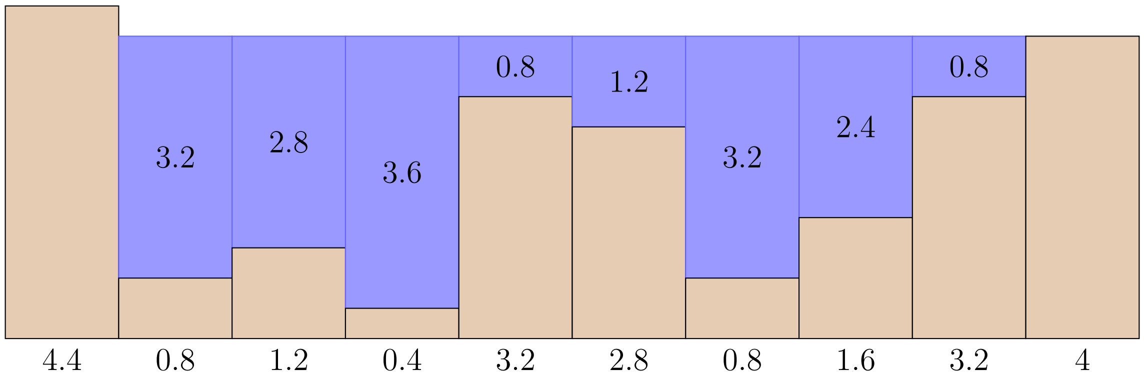


Example 3

`Input 10 heights:` `4.4 0.8 1.2 0.4 3.2 2.8 0.8 1.6 3.2 4`

`Amount of water: 18`

Example diagram:



Example 4
```
Input 10 heights:  0.4 0 1.2 0.4 3.2 2.8 0.8 1.6 3.2 4
Invalid heights!
```

Example 5
```
Input 10 heights:  -0.4 0 -1.2 0.4 3.2 2.8 0.8 1.6 3.2 4
Invalid heights!
```

Additional Comments, Clarifications, and Hints:

- One of the major difficulties in this problem is simply making sure you understand what you are being asked to do, and how we have asked you to do it. **Do not start programming if you do not understand the problem**. Programming without fully understanding the goal often leads to buggy code that is very hard for you to fix.
- The functions we have asked you to implement drastically simplify the task of developing this program. Make sure you understand these functions and how they fit into the design.
- With careful, incremental development this program can be rather straightforward to program, however, without incremental development this can easily turn into a nightmare. We recommend the following implementation order for the easiest time: Start by implementing the simpler functions `leftHeight` and `rightHeight`. Once these are done write a main program that tests them directly and make sure they work as expected. Once they are perfect delete the testing main program and move on to `waterHeight`. Only begin writing the main program when you are positive the others work as expected.

When you are done, name the source code file <username>_5A.cpp (where you replace <username> with your U of M email address) and submit it using the HW 5 Problem A submission link on the class website. Remember to follow the naming convention diligently.

Problem B: A Modified Metal Plate (30 points)
Many science and engineering simulations have the general form of the metal plate problem from Lab 6: you have a number of sample sites in two or three dimensions and run calculations on them over time until some stopping condition is achieved. This problem asks you to revisit that lab problem. Specifically, it asks you to make the following changes:

1. Your program should work with any grid size up to 10 rows by 10 columns. To do this, declare the array sizes to be 10 rows and 10 columns, but also keep track of the number of rows and number of columns that are actually used. So, for example, if the number of rows and columns used is 3 and 5, respectively, then the program should just use the first three rows and first five columns of the arrays, and ignore any other rows and columns.

2. Instead of having input from a user, have most of the input from a file. Specifically, the user should input two items: (a) a string that is the name of the data file to use, and (b) the tolerance. The initial plate temperatures will be in a data file that has the following format: The first line contains two positive integers, the first being the number of rows for the grid, the second the number of columns. The next line contains the first row of temperature data, and has as many numbers are there are columns. The following line in the file contains the next row of temperature data, and so on. An example of a data file is shown below. Note the number of rows does not need to equal to number of columns. Also note that, in contrast to the lab problem, each location in the initial grid might have a different temperature. For example, the first (top) row isn't required to have all its elements having the same temperature.

3. If the program cannot open the data file, then your program should print a message "Cannot open file" and terminate using `exit(1)`. If the user inputs a 0 or negative tolerance, then your program should print a message "Tolerance must be positive" and terminate using `exit(1)`. However, you may assume the data in the file is all valid, so you do not need to validate it.

4. Use a different stopping condition. Specifically, write a function `computeDiff` that calculates and returns the quantity

   $$\sqrt{\sum_{i=0}^{NROWS-1} \sum_{j=0}^{NCOLS-1} (T_{ij}^{(new)} - T_{ij}^{(old)})^2}$$

   That is, find the square root of the sum of the squares of the differences over all array locations. (Note in the formula above NROWS and NCOLS represent the number of rows and columns *actually used*.) Your program should stop when this quantity is less than the user-input tolerance.

5. After the program satisfies the stopping condition, print out the results as a grid, with each number printed in a field of width 9 and with 3 digits to the right of the decimal point.

Here is an example file:

```
3 4
1.0 1.0 1.0 2.0
1.1 1.2 1.2 2.0
1.2 1.2 1.2 2.5
```

And here is an example of running the program on that data file:

```
Input the data file name:  hw5btest1.dat
Input the tolerance:  .1

    1.000    1.000    1.000    2.000
    1.100    1.163    1.331    2.000
    1.200    1.200    1.200    2.500
```

Additional Comments, Clarifications, and Hints:

- As usual, construct the program incrementally.
- There will be a good amount of code for this problem. However, if you constructed a good solution for the related lab problem, this homework problem, although not easy, should not be extremely difficult.
- Like the lab problem, you might find it very useful to print out intermediate results when developing your program. However, make sure you remove any such extra output before you turn in your program.
- As usual, come up with additional test cases on which to test your program.
- Note the user should be able to enter any data file name, not just the name in the example above.
- Even though you might have worked on the lab problem with a partner, your work on this homework problem should be individual. More specifically, you are welcome to reuse a solution to the lab problem that you and a partner co-developed. However, you must modify that code to create a solution for this homework problem *on your own*.

When you are done, name the source code file <username>_5B.cpp (where you replace <username> with your U of M email address) and submit it using the HW 5 Problem B submission link on the class website. Remember to follow the naming convention diligently.