Computer Science II — CSci 1200
Lab 2

Overview

This lab has two parts. The first is focused on finding and fixing errors in your code. The second is focused on practice in thinking about and solving a simple programming problem.

The same rules apply here as in Lab 1. In particular, you may not use (a) email, (b) IM, or (c) any electronic resource other than your computer. After you download the program for Checkpoint 1, you may not use the internet at all.

Checkpoints

1. Using a web browser, download the file `diagonal_error.cpp` from

   `http://www.cs.rpi.edu/academics/courses/fall04/cs2/lab02/diagonal_error.cpp`

   This is a buggy version of a program that is supposed to produce the following output (we saw similar programs in class):

   ```
   What is your first name? george

   **********
   *        *
   * george *
   * georg  *
   * geor   *
   * geo    *
   * ge     *
   * g      *
   *        *
   **********
   Press any key to continue
   ```

   Create a project containing `diagonal_error.cpp`, compile it, run it, and then look closely at the output.

   How would you go about finding and fixing the bugs in such a program? I use at least four methods, usually in combination:

- A careful reading of the code and the output, trying to mentally work through the logic of the code and where it might have gone wrong.
- Stepping through the code with the debugger to understand exactly what it is doing and why.
- Inserting `cout` statements at key points in the code to print important values and the contents of data structures. I make sure to end each of these debugging `cout` with an `endl` (see discussion in Lecture 1).
- Running the problem on very small examples that I understand.

In the case of this program, I would read the code very carefully, run it with a small input (such as just the input name "C"), and check how the sizes of the various lines of output are being computed. You can use the debugger if you wish, but I didn't find it terribly helpful on this example. We will learn more about the debugger in the future.

In the case of this program, read the code very carefully; why are the same letters printed in every output line? In particular, which part of the code is responsible for reducing the number of letters printed from one line to the next? Something is missing....

To complete Checkpoint 1, find and fix the errors. You must be able to explain to a TA where the errors were in the code and what their effect was.

2. The next two checkpoints address exactly the same problem. Here is a description of the problem:

> You are given a sequence of numbers (doubles) of unknown length and you need to find (a) the smallest gap between any two numbers and (b) the largest gap between any two numbers. For example, suppose the values are
>
> ```
> 1.1 89.2 37.2 98.3 39.3 66.7 -2.4
> ```
>
> Then, the numbers that are closest together (in value) are 37.2 and 39.3, so the smallest gap is $39.3 - 37.2 = 2.1$. The largest gap is $98.3 - (-2.4) = 100.7$.

This may sound like a silly little problem, but a slightly more sophisticated version of it involving point locations on a grid has practical applications in automated cartography.

Your job is to write two different programs to solve this problem.

The first, for checkpoint 2, can not use any sorting. Think about how you are going to read in and store the values, and then think about how you are going to check all pairs in order to find the largest and smallest gaps. If you have trouble thinking about finding both the largest and smallest gaps, start with the largest gap, and then handle the smallest gap.

Note: you may use the library function `double fabs(double)` to compute the absolute value of a `double`. You will need to include the header file `cmath.h` in order to use this function.

3. Now, rewrite your solution to this problem. (Use a different file.) Sort the sequence after you read it in. You can now find the largest gap instantly and trivially. Finding the smallest gap requires a little more work, but not much. Do not use the same logic that you used for your code in Checkpoint 2; you need something simpler.

Discussion

The second solution, if you write it correctly, should be faster and simpler. The difference in complexity between the two solutions would be even larger if you had to output the pair of numbers that is closest together and the pair that is farthest apart. This illustrates several important points in solving programming problems:

- Most problems can be solved in multiple ways. Don't always take the first solution you think of.
- Using standard library algorithms like `sort` can make programming problems easier. Even if you had thought of sorting the numbers, you wouldn't have implemented this solution if you had to implement the sorting function on your own.
- Transforming the data (often by sorting) can make some programming problems much easier.