Programming Project 8

EE312 Fall 2013

Due Nov 13, 2013

No late submissions accepted!

FIVE POINTS

General: In this project you will be writing (nearly) the entire program from scratch. There is no "main" function provided for you. There are a number of design choices you will have to make about what struct types you'll want to use, and what data structure(s) make the most sense for this project. Good luck!

Your Mission: You can write as many .h files as you wish. Your entire project, however, should be contained in the file Project8.cpp. When we test your project, we will only compile Project8.cpp – all of your functions must be in this file (or #included from this file).

Background: For this project, you've been hired by the FIA to write a timing and scoring application Formula 1 race in Austin. Your application will read input from a file named "input.txt". This file consists of zero or more pairs of integers. Typically, each pair will be on a line and the next pair will appear on the next line. The first number in a pair is the car number. The second number in a pair is a timestamp (in units of milliseconds). The line identifies the time that the corresponding car crossed the finish line on the race track for one lap of a multi-lap race. For example, if you see the number 42 followed by the number 1000, then that means that car #42 crossed the finish line 1000 milliseconds after the start of the race. Since during the race each car will run many laps, you will see car #42 cross the finish line many times. There will be one pair of numbers in the file for each lap that each car completes. Your challenge is to identify the finishing order of the cars (i.e., which car won the race, which came in second and so forth) and what was the fastest lap time for every car in the race.

Stage 1, Input: I've made some minor modifications to the readNum function that we used in Projects 3 and 6. The function now returns a true if the function successfully read a number from the input file and returns false if all the numbers have already been read. In that way, you can process all the input in the file simply by calling readNum until you the function returns false. If the function returns false, then it did not read any input. You also know that there are no more numbers in the file and it's time to produce your output.

As you are performing input, you will want to keep a running log of all the laps performed by every car. Eventually, you will need to know which car completed the most laps, at what timestamp each car completed its last lap, and how long each lap was for each car in the race. You're on your own for designing the data structures that will best allow you to keep track of this information. All we tell you is that the input file consists of pairs of numbers with no punctuation or letters in between. The numbers can be read by readNum, and the first number in each pair is the car number while the second number is the timestamp when that car completed its next lap. For this project, you can assume that all the timestamps are in order. For example, I won't tell you that car #42 completed a lap at time 100,000, and then later in the input file tell you that car #4 completed a lap at time 50,000.

All of the numbers used in this project are non-negative. Car numbers can be any value (the actual car number that your application uses is not the number painted on the car, but rather is a transponder ID code from a device mounted inside the car that communicates with the race track). You will not know what the car numbers are until you've processed the entire input file. It is possible that a car will start the race late, and you will see that car number in the input file only have other cars have completed one or more laps. Your project must work with any non-negative car number up to nine decimal digits (i.e., up to but not including 1,000,000,000).

All of the timestamps are in units of milliseconds and all the timestamps represent the amount of time that has elapsed since the start of the race. Note that when a car first crosses the finish line, that car is starting its first lap. Make a note of the timestamp so that when the car crosses the finish line for the second time you can subtract the time stamps and compute the time taken to complete the first lap.

Continue reading the pairs of numbers until you readNum returns false. Once readNum has returned false, you've reached the end of the input file and it's time to produce the output.

Stage 2: Output.

You should print out the number of cars that appear in the input file (i.e., the number of cars in the race).

The winner of the race is the car that completes the most laps. If two or more cars complete the same number of laps, then the winner of the race is the car that finished its last lap at the earliest timestamp. So, for example, if car #42 and car #5 each completed 50 laps, but car #42 completed its 50th lap at timestamp 4,000,000 while car #5 completed its 50th lap at timestamp 3,980,000, then car #5 won the race. Your first concern when producing output is to correct print the finishing order of the race. You must print the entire order – i.e., you must identify which car finished first, which finished second, and so on through the entire set of cars that competed in the race.

For each car, you must identify and print the fastest lap run by that car during the race.

Your output must use the following formatting.

```
There are 3 cars in the race
Position 1 was car 42, completing 60 laps with the best lap of 123.980 seconds.
Position 2 was car 5, completing 60 laps with a best lap of 124.002 seconds
Position 3 was car 1, completing 59 laps with a best lap of 127.250 seconds
```

Note: to force printf to display the times numbers you can use

```
     printf("%d.%03d", t / 1000, t % 1000);
```

where t is an integer and is the time in milliseconds. You'll probably find it is much easier to use integer variables with units of milliseconds rather than floating point variables for measuring time in this project.

Stage 3: Performance requirements

Your program must run in no worse than O(N log N) time complexity where N is the number of input pairs in the file input.txt (i.e., N is the number of lines in input.txt). You can (and should) assume that the car numbers are random, and the timestamps are in increasing order. You can use any of the data structures and/or sorting algorithms we've discussed in class to complete your project. You may want to use more than one. You can find implementations of our data structures (and sorting algorithms) in the examples directory on the repository.