

CSCI-1200 Data Structures — Fall 2020

Homework 2 — Tennis Classes

In this assignment you will practice using C++ classes as you parse and compute statistics from the results of *Grand Slam* tennis championships (the Australian Open, the French Open, Wimbledon, and the U.S. Open). *Please carefully read the entire assignment before beginning your implementation.*

Here's a crash course in tennis scoring: A *match* between two players consists of a number of *sets*. For men's Grand Slam events, the first player to win three sets is the winner of the match (a.k.a., the best of five). Each set in the match consists of a number of *games*. The first player to win six games is the winner of the set, but the player must win by two games. If the set score gets to 6-5 (or 5-6), the players play a twelfth game. If the set score is 7-5 (or 5-7), the set is over. If the players are tied at 6-6 they play a special game called a *tiebreak* and the winner of the tiebreak wins the set, 7-6 (or 6-7). At the Australian Open, French Open, and Wimbledon, there is a special exception and in the fifth set they do not play a tiebreak. They just keep playing regular games until one player is ahead by two games (resulting in some fantastic 5 hour matches!) More details on the tennis scoring system are available here: http://en.wikipedia.org/wiki/Tennis_scoring_system. As an example of the input format, here is the result of the men's final at the 2015 Australian Open:

Novak Djokovic d. Andy Murray 7-6 6-7 6-3 6-0

In this match, player *Novak Djokovic* defeated player *Andy Murray* in a four set match. Djokovic won the first set (which went to a tiebreak). Murray won the second set (also a tiebreak). In the third set Murray only won three games, and in the last set Murray did not win any games.

File I/O and Command Line Arguments

Your program will run with three command-line arguments. The first is the input file containing the match information. The second is the output file where you will write the computed statistics. The third argument will indicate which data table should be printed. Valid options for the third argument are: `--match_stats`, `--game_stats`, or `--custom_stats`. For example, here is a valid command line to your program:

```
./tennis_statistics.out sample_scores.txt sample_scores_out.txt --match_stats
```

We provide a variety of small and large input files, and sample output for some of these input files. The input data was originally downloaded from <http://stevegetennis.com/>, but the format has been modified to simplify your work to parse the input.

Each match is listed on a separate line. The winner is always listed first. Each player has a first name and a last name (two strings). The special string "d." is placed between the two names. Each set is a string concatenating the number of games the first player won with the number of games the second player won, with the character "-" between.

Statistics Collected and Output

When `--match_stats` is specified, your program should create a table with the players ordered by the **percentage of matches** they won. Alphabetize players by last name, then first name if they are tied. Each row of the table should include the player, the number of matches won, the number of matches lost and the percentage of matches won. You should do a little bit of nice formatting to this output (see the example code from lecture and look at STL `iomanip` library references).

For example, given an input file with these matches:

```
Marcos Baghdatis d. Radek Stepanek  6-4 6-3 3-6 0-6 7-5
David Nalbandian d. Danai Udomchoke  6-2 6-2 1-6 6-7 6-1
Marcos Baghdatis d. Ivan Ljubicic   6-4 6-2 4-6 3-6 6-3
Marcos Baghdatis d. David Nalbandian 3-6 5-7 6-3 6-4 6-4
```

Your program will produce this table:

```
MATCH STATISTICS
Player      W    L  percentage
Marcos Baghdatis  3    0    1.000
David Nalbandian  1    1    0.500
Ivan Ljubicic    0    1    0.000
Radek Stepanek   0    1    0.000
Danai Udomchoke  0    1    0.000
```

When `--game_stats` is specified, your program should create a table with the players ordered by the **percentage of games** they won. Again alphabetize players by last name, then first name if they are tied. As in the first part, each row lists the player, the number of games won, the number of games lost and the percentage of games won. Here's the output for the data above:

```
GAME STATISTICS
Player      W    L  percentage
David Nalbandian 49   44    0.527
Radek Stepanek   24   22    0.522
Marcos Baghdatis 73   69    0.514
Ivan Ljubicic    21   25    0.457
Danai Udomchoke  18   25    0.419
```

Finally when the `--custom_stats` option is specified, it is a chance for you to be creative. You will collect and output some other statistic from the matches. For example, you could identify which sets went to a tiebreak, and sort the players by performance in these tiebreak sets. Another example would be to find all the matches where the ultimate winner lost the first set and output the players who most often “come from behind to win”.

Extra credit will be awarded to particularly interesting statistics that require clever programming. An important task for this part of the assignment is to write a concise description (~ 100 -200 words) of your new statistic. Put this description in your `README.txt` along with any other notes for the grader. Be sure to tell the grader which dataset best demonstrates your new statistic. You may create your own dataset and include it and your program's output for that test case with your submission. Include interesting sample output from your custom statistic with your submission.

Useful Code

To control the formatting of your tables, you'll want to read up on the various iomanipulators: `std::setw(int)`, `std::setprecision(int)`, and `std::fixed`. And don't forget about the `sort` function that can be used to order the contents of a `vector`.

To help you parse a string from the input file that represents a set, we provide the following code to get you started:

```

// Parses a string that represents a set (i.e., "6-3") by breaking the
// string into two substrings and converting those strings to
// integers, which are returned via call-by-reference parameters
void parse_set(std::string &set, int &games_won, int &games_lost) {
    int i = set.find('-');
    games_won = std::stoi(set.substr(0,i));
    games_lost = std::stoi(set.substr(i+1,set.size()-i-1));
}

```

With the code above and the `>>` input stream function for strings used in lecture you can implement all of the parsing necessary for this assignment. *Hint:* By looking at the results of the first 3 sets, you can determine whether a 4th set was played and you should read in another set *or* whether the match was over after 3 sets and next string contained in the file is part of the next match. In fact, your code should not care or depend on the presence or location of newlines (or extra newlines or other whitespace) within the input file.

Initially, we recommend you focus your testing on the dataset of matches that required 5 sets. Once that is working you can extend your solution to handle all matches.

Program Requirements & Submission Details

Your program should involve the definition of *at least one class* that has its own `.h` and `.cpp` files, named appropriately. You should also continue your practice of STL `vector`, STL `string`, and STL input/output/file streams.

Be sure to read [“Good Programming Practices”](#) as you put the finishing touches on your solution. Use the provided template `README.txt` file for notes you want the grader to read. You must do this assignment on your own, as described in the [“Collaboration Policy & Academic Integrity”](#) statement. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.