CSCI-1200 Data Structures — Fall 2010
Homework 2 — Bowling Classes

In this assignment you will parse and compute statistics of the game of bowling. Scoring can get a bit confusing for newbies to the game (especially those who rely on automated scoring at the bowling alley), but we have summarized everything you need to know to complete this assignment. Please read the entire handout before starting to code the assignment. There are also many excellent reference webpages on bowling, for example: `http://en.wikipedia.org/wiki/Ten-pin_bowling`

The Basic Game

Here's a crash course on the game of bowling: Each game consists of 10 frames. For each frame, 10 pins are set up at the end of the alley. The player gets two chances to throw a ball down the alley trying to knock down as many pins as possible. If the player knocks down all 10 pins with the first throw, this is called a *strike*, and the second throw is not necessary for the frame. If the player knocks down 0-9 pins on the first throw, and all remaining pins on the second throw, this is called a *spare*.

The Scoring

The score is tracked cumulatively, with a running total noted per frame. If fewer than 10 pins were knocked down in the frame, the number of pins knocked down in this frame is simply added to the score from the previous frame. Thus, if a player does not achieve any strikes or spares during the entire game, the final score is just the total number of pins knocked down.

If the player completes a spare, that frame will receive a bonus score (in addition to the 10 pins knocked down in that frame) equal to the number of pins knocked down in the first throw of the following frame. Similarly, if the player achieves a strike, that frame will receive a bonus score equal to the number of pins knocked down in the next two throws.

The Tenth Frame

The tenth frame is a little special. If the player achieves a strike in the tenth frame, he or she gets a fresh set of pins and two bonus throws to earn additional points. If the first of these bonus throws also is a strike, he or she gets yet another fresh set of pins for the second bonus throw. If, on the other hand, the player achieves a spare in the first two throws of the tenth frame, he or she gets a fresh set of ten pins and one bonus throw to earn additional points.

The maximum possible score for a perfect game (10 strikes, followed by 2 more strikes on the bonus 10 frame throws) is 300.

Scoring Sheet Conventions

The most common way to chart and display the game score for one or more players, both during and after a game, is in a standardized table with 11 columns and one row per player. The leftmost cell of each row contains the player's name and the remaining 10 cells display the results for each frame. The scores for each of the throws of the frame appear in the top half of the frame and the running total score appears in the bottom of each frame. If a strike was achieved an '`X`' is marked in the upper right corner of the frame box. If a spare was achieved in the frame, the number of pins knocked down in the first frame is marked in the upper left corner of the frame box and a '`/`' is marked in the upper right corner of the frame box. If fewer than 10 total pins were knocked down, the number of pins for each the two throws are simply noted in the left (first throw) and right (second throw) top corners of the box. If no pins were knocked down in a throw, a '`-`' is marked rather than a '0'. Because the tenth frame is a little different, with up to 3 throws, the last cell in the row is a little wider to present the results of all of those throws. See the examples below.

Input Format

You will parse an input file with the scores for one or more games of bowling (not in any particular order). Here is a sample input file:

```
George Smith 9 1 10 8 2 10 8 2 9 1 10 8 1 10 10 10 2
Sally Jones 10 10 10 10 10 10 10 10 10 10 10 10
Fred Adams 2 0 1 5 6 4 10 2 2 0 0 0 8 2 8 2 3 0 2
Betty Smith 7 2 8 2 8 1 8 2 10 8 1 9 1 10 7 2 1 9 10
```

The file contains a series of games, one per line. Each game begins with two strings that represent the player's first and last names and is followed by a set of integers that represent the number of pins knocked down by each of their throws during the game. Note that the number of throws can vary, depending on how many strikes or spares were achieved during a game. A game with no strikes or spares will contain exactly 20 throws (2 throws per frame). A perfect game (score 300) will contain exactly 12 throws (1 strike per frame plus 2 bonus throws). You may assume that there no are errors in the input file and that there are no anomolies in the game play (all games are complete, no fouls, etc.). To verify that your parsing is correct we suggest that you perform simple logic checks in your code (e.g., make sure that the total number of pins knocked down in each frame is <= 10). You should *not* assume any specific formatting of the white space within the line (there may be extra whitespaces or tabs or newlines throughout the file). We *strongly recommend* that you use the `iostream >>` operator to parse this file and not use `getline` or other C-style file I/O commands.

File I/O and Command Line Arguments

Your program will run with two command-line arguments, one being the name of the input file described above and the other being the name of the output file where you will write a nicely formatted score table and a couple of interesting statistics (described below). Example input and output files posted on the course website. For example, here is a valid command line to your program:

```
bowling_scores.exe 2010_US_Open.txt out_2010_US_Open.txt
```

We have provided you with several bowling sample datasets, including data from The Professional Bowlers Association website: `http://www.pba.com/`. The format has been modified to ease parsing.

Statistics Collected and Output

The output will be in *three parts*. First is an ASCII art table mimicking the standard bowling scoresheet described above. The table contains all of the games in the input file, sorted alphabetically by player last name (and then first name if the last names are the same). If the input file contains multiple games by the same player, all games will be included in the table, but the exact order of those games in the final table is not specified. To ease grading, please exactly follow the format of the table below. The width of the first column should be automatically adjusted to fit the player with the longest name.

```
-------------------------------------------------------------------------------
| Fred Adams   |  2 - | 1 5 | 6 / |   X | 2 2 | - - | - 8 | 2 / | 2 3 | - 2   |
|              |    2 |   8 |  28 |  42 |  46 |  46 |  54 |  66 |  71 |    73 |
-------------------------------------------------------------------------------
| Sally Jones  |    X |   X |   X |   X |   X |   X |   X |   X |   X | X X X |
|              |   30 |  60 |  90 | 120 | 150 | 180 | 210 | 240 | 270 |   300 |
-------------------------------------------------------------------------------
| Betty Smith  |  7 2 | 8 / | 8 1 | 8 / |   X | 8 1 | 9 / |   X | 7 2 | 1 / X |
|              |    9 |  27 |  36 |  56 |  75 |  84 | 104 | 123 | 132 |   152 |
-------------------------------------------------------------------------------
| George Smith |  9 / |   X | 8 / |   X | 8 / | 9 / |   X | 8 1 |   X | X X 2 |
|              |   20 |  40 |  60 |  80 |  99 | 119 | 138 | 147 | 177 |   199 |
-------------------------------------------------------------------------------
```

The second portion of the output lists all the games sorted by final score, but rather than displaying the full details, it only lists the final game score:

```
Sally Jones   300
George Smith  199
Betty Smith   152
Fred Adams     73
```

The third and final part of the output is a chance for you to be creative. Brainstorm a new interesting statistic that can be calculated and presented from this data. Examples include sorting the games by the total number of strikes scored per game, or counting how many times the during the game the player throw a "gutter ball", knocking down no pins.

Extra credit will be awarded to particularly interesting statistics that require clever programming. The most important task for this part of the assignment is to write a concise description (< 100 words) of your new statistic. Put this description in your *plaintext* `README.txt` file along with any other notes for the grader. Be sure to tell the grader which dataset best demonstrates your new statistic, and include a sample of the output. Feel free to create your own dataset and include it with your submission.

Useful Code

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

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. We have provided a series of series of datasets to aid in your program development and debugging. The first data set `no_strikes_or_spares.txt` is the simplest, because each game consists of exactly 20 throws and the score calculation is simple. We highly recommend you first focus on reading this file and produce the full and correct output for this data. The majority of points for this homework assignment will be awarded for programs that work correctly for the simple examples. Once that is complete, you may tackle the scoring complexities for games with strikes, spares, and bonus throws in the tenth frame.

Do all of your work in a folder named `hw2` inside of your Data Structures homeworks directory. Use good coding style when you design and implement your program. Be sure to make up new test cases and don't forget to comment your code! Please use the provided template `README.txt` file for any notes you want the grader to read. **You must do this assignment on your own, as described in the "Academic Integrity for Homework" handout. If you did discuss the problem or error messages, etc. with anyone, please list their names in your `README.txt` file.** When you've finished writing, testing, debugging, and commenting your code, prepare and submit your assignment as instructed on the course webpage. Please ask a TA if you need help preparing your assignment for submission or if you have difficulty writing portable code.