CSCI-1200 Data Structures — Fall 2012
Homework 2 — Football Classes

In this assignment you will parse results data from and compute statistics of the game of NFL football. The full rules of the game are overwhelming, but we have summarized the basics you need to know to complete this assignment. Please read the entire handout before starting to code the assignment.

The Basic Game

Here's a crash course on the game of football: Two teams meet for the game, in one of the team's home stadiums. The teams are thus designated as the *visiting team* and the *home team*. Each game consists of 4 15-minute quarters. A different number of points are earned for several different plays: 2 points for a *safety*, 3 points for a *field goal*, 6 points for a *touchdown with no extra points*, 7 points for a *touchdown with a successful extra point kick*, and 8 points for a *touchdown with a successful 2 point run or pass*. Field goals and 7-point touchdowns are the most common plays. 6- and 8- point touchdowns occur occasionally, and safeties are quite rare. If the score is tied at the end of the 4th quarter, the game continues into overtime in an attempt to determine a winner.

Input Format

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

```
Sunday     September  12 2010
  Falcons      0  3  3  3  0    9
  Steelers     3  0  3  3  6   15
Sunday     September  26 2010
  Falcons      7  7  3  7  3   27
  Saints      14  0  7  3  0   24
Sunday     October    31 2010
  Titans       5 14  0  6  0   25
  Chargers     7  7 10  9  0   33
Monday     December   27 2010
  Saints       3  7  0  7  0   17
  Falcons      0  7  0  7  0   14
```

The input file contains a series of games, which may span multiple lines, but you should not depend on any particular newline or white space layout. There will be no punctuation in the file. Each game begins with the date (day of the week, month, date, and year), then the visiting team's name and game scores and then the home team's name and game scores. The game scores consists of 6 integers: the points for each of the 4 quarters, the points scored in overtime, and the total points for that team in the whole game. If the score was not tied at the end of the fourth quarter, no overtime was necessary, so a zero is recorded for both teams in the overtime slot.

Note that for some quarters we can determine with 100% certainty the plays that occurred. For example, a quarter total equalling '3' is a single field goal, and '5' is a field goal plus a safety. However, some scores are ambiguous: e.g., '9' could be 3 field goals or one field goal and one 6-point touchdown, and a '7' could be 1 field goal plus 2 safeties. Note that for the latter example we can in fact quite confidently state that it is a touchdown, because safeties are exceedingly rare.

You may assume there are no errors in the formatting of the input file. But we suggest that you verify that your parsing is correct by performing simple logic checks in your code (e.g., make sure that the game score total is the sum of the 5 preceeding integers). You can also verify that the components of the date are legal (e.g., the date does not exceed the maximum days for that month). 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 `getc` 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 nicely formatted tables for several statistics (described below). Example input and output files are posted on the course website. For example, here is a valid command line to your program:

```
./football_scores.exe 2010_nfc_east.txt out_2010_nfc_east.txt
```

We have provided you with several sample football datasets, including data from the ESPN NFL Scoreboard website: `http://scores.espn.go.com/nfl/scoreboard`.

Statistics Collected and Output

The output will be in *three parts*. First is a listing of all the games in the input file, sorted by *total point differential*, with the games with the smallest differential (closest total score between the two teams) first. The visiting team should be listed first, with a short message indicating whether they defeated or lost to the home team. If two or more games have the same differential, the game with the higher total number of points should be listed first. If two or more games remain tied, then they should be listed alphabetically by visitor name, and if a tie still remains, then alphabetical by home team name.

To ease grading, please exactly follow the format of the table below and in the provided examples. The width of the team name columns should be automatically adjusted to fit the team with the longest name.

```
ALL GAMES, SORTED BY POINT DIFFERENTIAL:
Falcons   defeated  Saints    27 - 24
Saints    defeated  Falcons   17 - 14
Falcons   lost to   Steelers   9 - 15
Titans    lost to   Chargers  25 - 33
```

The second portion of the output lists all the teams sorted by win percentage (number of games won / total number of games played). If two teams have the same win percentage, the team that played the higher number of games should be listed first. If two teams remain tied, then the teams should be listed alphabetically. To ease grading, please follow the format of this table exactly.

```
ALL TEAMS, SORTED BY WIN PERCENTAGE:
Chargers   1 win(s) -  0 loss(es)  1.00
Steelers   1 win(s) -  0 loss(es)  1.00
Saints     1 win(s) -  1 loss(es)  0.50
Falcons    1 win(s) -  2 loss(es)  0.33
Titans     0 win(s) -  1 loss(es)  0.00
```

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. Your statistic must make use of the scores by quarter (e.g., "which teams are better at coming from behind?", or "who scores the most points during the 4th quarter?") *or* the dates the games were played (e.g., "which teams perform better on Monday nights than Sunday nights?") *or* make an educated deduction of the number of field goals and touchdowns in each game. For example, the statistic below attempts to identify the teams with the best defense, defined as fewest average touchdowns (6, 7, or 8 point) by their opponents. The format of this third table depends on your choice of statistic and thus is not specified.

```
ALL TEAMS, SORTED BY AVERAGE OPPONENT TOUCHDOWNS PER GAME:
Steelers   0 touchdown(s) in  1 game(s)  0.00
Falcons    6 touchdown(s) in  3 game(s)  2.00
Saints     5 touchdown(s) in  2 game(s)  2.50
Chargers   3 touchdown(s) in  1 game(s)  3.00
Titans     4 touchdown(s) in  1 game(s)  4.00
```

Extra credit will be awarded to particularly interesting statistics that require clever programming — for example, a recursive implementation to deduce the number of field goals and touchdowns in a game. 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`, `std::right`, etc. And don't forget about the `sort` function, which 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 datasets to aid in your program development and debugging.

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.