Problem Set 2

Due before midnight on Thursday, February 18, 2016.

1 Assignment Goals

1. Produce an application with more than one class, appropriately split into multiple files.
2. Learn the CSV (comma-separated values) file format that can be produced by many spreadsheet programs, and learn how to read and parse a CSV file.
3. Learn about class `vector<T>` in the standard library.
4. Learn about functions `sort()` and `stable_sort()` in the standard library.
5. Learn more about I/O, including the string version of `getline()`, class `istringstream`, manipulator `setw()`, and when and how to read past a delimitor.
6. Practice proper I/O error checking and eof handling.
7. Write a print function using delegation.
8. Learn how to pass a C-style `char*` string to a function with a C++ `string` parameter.
9. Learn how to use range-based for-loops.

2 Multiparty Decision-Making

A commonly arising situation is for a group of people to make a collective decision that somehow reflects the group's best judgment. A familiar example is the election of a board of directors for an organization. Here, there is a collection of candidates and some number v of vacant seats. The voters collectively choose a subset of size v of the candidates to fill those seats. Another example arises in figure skating competitions. Here, each judge assigns a score to each skater. The judges marks are then combined to rank the skaters and award medals.

The two problems are similar but not identical. In both cases, each voter/judge express an opinion about each candidate/skater. In the first, the goal is to determine the subset of candidates to fill the vacant seats. In the second, the goal is to rank the skaters from top to bottom.

3 The 6.0 Scoring System

In the next few problem sets, we will implement a version of the 6.0 scoring system that was used for international figure skating competitions until 2005 and is still used in some non-qualifying competitions. It's history is described on the Wikipedia 6.0 system page.

In an event, each skater skates according to the requirements for the event in front of a panel of judges. Each judge is given a scoring sheet with the names of the skaters listed in order of skating. The judge assigns each skater a mark in the range $[0.0, \ldots, 6.0]$ that reflects the quality of the performance, with 6.0 being the best.[1] At the end of the event, the scoring sheets from each

[1] This is a slight simplification for how each judge's mark is actually obtained.

judge are collected and given to the accountant, who then combines the marks according to the 6.0 ranking algorithm to produce the final results.

A nice description of the somewhat complex ranking algorithm can be found here. We will be implementing it in a subsequent assignment.

4 Problem

This assignment will focus on one aspect of implementing a 6.0 scoring system: How to represent the input data, both externally in a file and internally in appropriate C++ data structures. Here we will simplify the problem by focusing on a single event.

We assume that the marks from the judges' scoring sheets have been entered into a spreadsheet with one line per skater and one column per judge. The first line of the file contains column labels. The first column contains the skaters' names. Thus, if there were 20 skaters and 7 judges, the spreadsheet would have a total of 21 rows and 8 columns.

We assume further that the spreadsheet has been saved as a CSV (comma-separated values) file. A CSV file is a text file with one line of text per row of the spreadsheet. The entries in each row are separated by comma (',') characters. Sample good and bad data files and sample output are available on the Zoo in `/c/cs427/assignments/ps2`.

Your problem is to write a C++ program that takes as a command-line argument the name of a CSV-format marks file and uses the data in it to contruct data structures that will be used in a future assignment by the ranking algorithm.

More specifically, your program must define two classes:

1. An object of `class Score` represents a single judging mark. It should have three private data members: the skaterID and judgeID, both of type `unsigned`, and the mark itself (a number in $[0.0, \ldots, 6.0]$, represented by a `float`).

   The class should have a constructor that initializes all three data members from its corresponding three arguments. In this way, a `Score` object is fully defined when first created.

   For now, the only other required `Score` function is `print()`, which should print a human-readable representation of the data in `Score`. It should *not* add a new line character at the end. Also, `operator<<()` should be defined as an inline function following the class definition as described in lecture 5.

2. `class SixOh` is the driver class and will be instantiated only once in `main.cpp`. It should have four private data members: the name of the input file represented by a `string`, a list of skaters and a list of judges, both represented by variables of type `vector<string>`, and a list of scores, represented by a `vector<Score>`. You may also choose to include other data members to use for communicating data among your member functions.

   The `SixOh()` constructor receives the name of the CSV input file as its argument and saves it to the file name data member.

   Member function `import()` should open the marks file whose name was passed to the contructor when the `SixOh` was instantiated. Next it should read and parse the line of column headings, checking that the first field contains the word "Skater", and reading in the column headings for the judges. This also determines how many judges are present. Finally, it should read each of the skater lines. The first field is the name of the skater and should be put in the skater list. Each remaining field contains the mark of the corresponding judge. The mark should be read and a `Score` object created and put into the lists of scores.

The other function you need for this assignment is `sortEvent()`. After `import()`, the `Score` objects will be ordered, first by skaterID and then by judgeID. `sortEvent` should sort the scores so that they are ordered first by judgeID and then by skaterID.

As usual, you should define a `print()` for the class. It should print out, appropriately labeled and formatted, the list of skaterID's, the list judgeID's, and the list of scores. `Sample output` for `marks.csv` show one reasonable way of presenting this data. Also `operator<<()` should be defined as an inline function to allow the printing of a `SixOh` object using the << syntax.

5 Programming Notes

1. The file `main.cpp` should have the same form as for PS1. The `run()` function should check the number of command line arguments furnished, obtain the file name, and create an instance of `SixOh` initialized by the file name string. It should then call the `SixOh` `import()` function to read the file and create the `Score` objects.

2. The print function for class `SixOh` should delegate the printing of a `Score` object to the print function in class `Score`.

3. To make things line up nicely, use the `setw()` manipulator to set output field widths.

4. To sort the scores, use `std::sort(` or `std::stable_sort()`. Define `operator<()` in the `Score` class to specify the new sort order. You will need to put `#include <algorithm>` in your header file.

5. A `const char*` argument (such as `argv[1]`) can be passed as an actual argument to a function that takes a a `const string&` parameter. This avoids an unnecessary copy operation. Rather, when the function is called, the reference parameter name is bound to an anonymous string that is constructed from the `const char*` actual argument. This is a clean way to convert a C-style string to a C++ `string`.

6. `vector<T>` is a template type. To use it, you replace `T` with the type of the elements you will be storing in the vector. Vectors support many many operations. For this assignment, you will only be using `push_back()` and `size()`. You will need to put `#include <vector>` in your header file.

7. You should be using vectors of `string` and vectors of `Score`, not pointers. In fact, there is no need for explicit pointers or `new`, and they should not be used in this assignment.

8. To print all of the elements of a vector, you need to iterate through the vector `v`. One could do this with an integer subscript going from 0 to `v.size()-1` and then use subscript to fetch the elements. However, I want you to use the range-based for loop. (See range-based for loop under statements and flow control.)

9. To read a line from a text stream `in` into a string `str`, use the string version of `getline()`, i.e., `getline(in, str)`.

10. To parse a line of text after reading it into a string `str`, create an `istringstream` input stream object `sin`, initialized from `str`. The new stream `sin` can be read just like any other input stream such as `cin`. When an attempt is made to read beyond the end of the string, `sin.eof()` becomes true.

11. When reading a comma-delimited text field, you can use `getline()` by specifying `','` as the third argument. This will read the text up to the comma-delimitor and then discard it.

12. When reading the numeric mark fields, reading will stop when it hits the comma delimitor, but it will leave the comma in the stream. You will have to explicitly remove the comma in order to read the next mark.

13. You should check that the number of marks read for each skater equals the number of judges, and that no marks are left unread at the end of the line. If the number of marks is wrong, or for any other file format error you discover, you can report a fatal error.

14. SkaterID's and judgeID's should be assigned in order, beginning with 1.

6 Grading Rubric

Your assignment will be graded according to the scale given in Figure 1 (see below).

#	Pts.	Item
1.	5	A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++11`.
2.	5	Running `make` successfully compiles and links the project and results in an executable file `scorer`.
3.	30	Running `scorer` on `marks.csv` produces correct output.
4.	10	Running `scorer` on the files `badmarks1.csv`, `badmarks2.csv`, ... results in the reporting of fatal errors.
5.	10	Each function definition is preceded by a comment that describes clearly what it does.
6.	30	All of the insructions in sections 4 and 5 are followed.
7.	10	All relevant standards from PS1 are followed regarding submission, identification of authorship on all files, and so forth.
	100	Total points.

Figure 1: Grading rubric.