11: Keeping Score

CSCI 6626 / 4526 April 2018

1 Goals

- To use an STL class and its iterator.
- To use input and output files.

2 The Permanent Scoreboard

During the game, each player has a scoreboard that records the columns he has won. At the end of the game, or when the player resigns, we want to record this data on a permanent scoreboard. For this purpose, we need two more classes: Score (the stats for one player) and ScoreBoard (a collection of all the players' stats plus the column statistics).

Score class. Every player will have a Score object. Its parts are:

- An array of 13 integers (not a vector). The first two slots of this array will record the number of games this player has played and the number of games he has won. Slots 2..12 will record the total number of times he has captured each column.
- A print function that displays all of the Score data, with a matching `operator <<` extension.
- void update (int, const int*)
  The parameters are the number of columns a player has captured (an integer 0..3) and a pointer to the Player's array of captured column numbers. This function must update the number of games played, games won, and column totals.
- A default constructor that initializes all 13 slots of the array to 0.
- A serialize function with an ostream& parameter. Write all 13 column totals to the ostream, separated by spaces and followed by a newline.
- A realize function with an istream& parameter. Read all 13 totals from the istream and store them in the totals array.

A notational shortcut. When using the ScoreBoard class, your code will often need to use one of these phrases:

```
map<char*, Score>
pair<char*, Score>
```

They are not nice to type and even worse to read. To make the code more readable and less error prone, you should put these two #define statements at the top of ScoreBoard.h:

```
#define SBoard   map<char*, Score>
#define SBPair   pair<char*, Score>
```

This will permit you to use the names SBoard and SBPair instead of the long, awkward, phrases.

Another shortcut. You may use the STL type string instead of a char array for the player's name, and instead of `char*` in the map and pair types.

3 The ScoreBoard class.

Derive this class from SBoard, that is, from the instantiation of the STL map class with a char* for the key field and a Score for the value. Define these members:

At the top of the .hpp file, `#define INF` to be the name of the file that contains the permanent scoreboard. Define `OUTF` to be the name of a file for output. In a real program, you would rename the input file with a name that indicates it is a backup, and give the output the same name as the normal input. There is no time in this term to worry about file renaming. Therefore, to test your program and demonstrate that the permanent scoreboard grows and changes, you will need to do the file-renaming manually.

- A constructor. Create one SBpair and insert it into the map. The key of this pair should be "Game", and the value should be a default Score. Your class will use this to record the total of all games played, all games won, and all times a column has been captured by anybody.
- A function named realize() that will open an ifstream and read in the file of score information. Call this function from the ScoreBoard constructor. To do this job, you need a local Score variable (declared inside the read loop, not created with new). Read the name from the file, then delegate the rest of the input to Score::realize. Make an SBPair out of the name and Score, and insert it into the map. When you insert this Score into the map, a copy of the data will be made, so you can safely re-use the local variables for the next line of input. Options:
  - Use a variable of type string to store the name, and use `<<` to read it from the stream.
  - Put the realize code in your constructor instead of in a separate function.
- A function named serialize() that will open an ofstream and write out the scoreboard. Call this from your destructor. (Option: put this code in your destructor.) To do this job, you need a local iterator variable. Iterate through the map, and for each pair, write the name to the output file and call Score::serialize() to do the rest of the job.
- A print function that prints both the key and the value part of every pair in the collection. You will need an iterator for the base class (SBoard). If you call your iterator k, then you can use it these ways to access the data in the map:

```
k = this->begin()  // sets the iterator to the beginning of the collection.
k != this->end()   // true if the k has not passed the end of the collection.
k->first        // is the key part of the pair that k refers to.
k->second       // is the Score part of the pair that k refers to.
```

- A function void update(const char* Name, const int* columns). This will be called each time a player resigns from the game or wins the game. The parameters are the player's name and the array of columns he has captured in this game. First, update the overall game statistics in the map. Then use the player's name to search the map and find the permanent scoreboard for that player. Increment the appropriate fields in this scoreboard. If your iterator is named k, then you should write `k->second.update( ... );` to call the Scoe::update function. This call will change the data in your map. You do not need another step to reinsert the changed data.

4 Modifications to Previous Classes

- In Game, add a member of type ScoreBoard. Modify your Game functions to update the scoreboard when a player wins or resigns and when the game is over.
- Add "Show Scoreboard" to the menu of options in oneTurn. Show the overall Game scoreboard and the scoreboard for the current player. Then permit the player to take a normal turn.
- Use type string instead of a char array in the Game class to read in the Player's name. (Easy.)
- Add a static class variable to the Player class and increment it every time a new Player is created. Add an accessor function to return the current number of players. Use this in the Game class to limit the number of players who join the game.