Computer Science II — Homework 4 — Song Lists

This assignment is due Thursday, February 22 at 11:59:59pm and is worth 80 points toward your homework grade. It requires that you have an understanding of lists and iterators, as covered in Lectures 8 and 9 and in Lab 5.

Your job is to keep track of the songs posted on the internet and (legally, of course) available for download and the people who have downloaded the songs. Sites on the internet will announce that a song is available and announce the download time required. Individuals will request songs, downloading them from the site for which the current download speed is the fastest. A site may go off the internet, in which case all of its songs must be removed and "forgotten" by the system. In other words, if the site came back on-line, none of the songs would be remembered by the system. At the end of the input, your program must print summary information about the songs and about individuals who have downloaded songs.

Initially, the lists of songs and individuals are both empty. The input is relatively simple. It consists of three different "requests". The first is

```
announce site-id song-name download-time
```

where `site-id` is a string giving the name of a site on the internet, `song-name` is a single string giving the name of the song, and `download-time` is a `float` giving the time in seconds to download the song. For example,

```
add 12adf4 beethoven-symphony-5 125.1
```

indicates that site `12adf4` has song `beethoven-symphony-5` available for downloading in 125.1 seconds. No output is necessary in response to this input. If the song is already available from this particular site, then replace the download time with the new time listed.

The second request is

```
download person song-name
```

where `person` and `song-name` are each single strings. For example,

```
download george-washington beethoven-symphony-5
```

indicates that `george-washington` wants to download `beethoven-symphony-5`. At this point, the program should find the site that has the fastest download of `beethoven-symphony-5` available. If no site is available then the program should output

```
beethoven-symphony-5 unavailable
```

Otherwise, the system should output

```
beethoven-symphony-5 downloaded by george-washington from 12adf4
```

(if `12adf4` is the site for which the fastest download is available). If the person has already downloaded the song, then download the song again anyway.

The third request is

```
remove site-id
```

which indicates that all songs for the given `site-id` should be removed. When removing a site for a particular song, if the removed site is the last site for the song, the song should be removed from the list of available songs.

At the end of the input, the program should output a blank-line and then output the songs (in alphabetical order) and for each song, all of the sites for which the song is available. (Assume without checking that there is at least one song.) The output of the sites for a song should be in order of download time, with the shortest download time first (lexicographic order in case of ties). The form of the output should be the name of the song, unindented, then the ordered output of all the sites, each on a separate line, with four spaces of indentation before the site-id and then one space before the speed.

The last part of the output should be a listing of the people and the songs they have downloaded. The order should be lexicographic by people and for each person the order the should be lexicographic by song. The person should be output on one line, unindented, and the songs should follow on separate lines, each indented by four spaces. (Sorry for being so picky about spacing, but being specific will help everyone during grading.) In case a person has no songs downloaded (s/he requested one or more songs that were not available and requested no songs that were available), output the line

```
<empty>
```

(four spaces before the `<`) instead of any songs. If a person has downloaded the same song more than once, the song should only appear in this output list once.

Here are some last, crucial details:

- **Your program should run with two command-line arguments**, the first being the input file and the second being the output file.
- **You must use lists** in this assignment and no vectors. Otherwise, you may use any technique from Lectures 1-9.
- Remember that the `std::list` class has its **own sort function**. You may pass a comparison function to this class. You may or may not need the sort function, depending on how you structures your program.
- There will be no formatting mistakes or spelling mistakes in the input. If two song names differ by one character, even if it is upper case vs. lower case, the songs are different. The same is true of sites.
- You will need to understand how to use the `insert` and `erase` member functions of the `std::list<T>` class. These are covered in Lectures 8 and 9.
- You will likely need a number couple of classes, with one of them being a Song class. This Song class will need to keep a list of sites that hold the song and the associated download speeds. (Hint: maintain the songs in order of download speed.) This implies that you need a very simple, "light-weight" class to hold these two pieces of information together. This is one case where forming a simple **`struct` is allowed**, as long as the struct is not used outside the Song class. Of course, the Song class must be a real class. To do all of the most effectively, you might want your Song output (at the end) made by a member function of the Song class. If you do this, please remember to **pass the output stream object by reference.**

- **Your submission** should be a zip file containing all of the header and source code files, together with a `readme.txt` file. A template for this will be posted on-line and it will require you to answer a question about the order notation ("complexity") analysis of your program.