CS 240 Program 2: Fall 2015
Simple Internet Radio Simulator
Assigned November 16, 2015

Think about how Pandora Radio might store music and select songs to play for users. Somehow, Pandora must continuously select and play a "next song" from a pool of appropriate songs. The listener then has the opportunity to click "Thumbs Up" to indicate that she likes the song, "Thumbs Down" to indicate that she does not, or neither. This information is then factored into the algorithm that selects subsequent songs. The system might learn that some user has an absolute favorite song (because she clicks "Thumbs Up" every time it plays), but even so, that song should not be played over and over; the fact that a song just played should reduce the chances that it plays again very soon.

For this program, you will implement a very simplistic version of a Pandora Radio song selection algorithm. Your algorithm will consider only the two factors mentioned above, namely (a) how recently the song was played, and (b) the user's feedback after previous plays of the song.[1]

Your program will read and respond to "commands" from standard input. These commands will allow users to add songs to a music library, and *simulate* (a) the playing of songs, (b) periods of time when the user is inactive, and (c) user feedback in the form of "likes" and "dislikes." To be clear, you will store information about songs (including the Song Title and duration), but songs themselves; you will simulate playing songs by advancing a virtual clock by the duration of a song, but you will not actually play the song.

1. INPUT

Your program should accept and respond to the following commands:

`RUN <filename>`

This command should cause your program to read and respond to all of the commands in the file named `<filename>`. That file may have any of the commands described below, but will not contain another "`RUN`" command.

`INIT <Time> <MaxSongs>`

This command sets the current simulated time to `<Time>`, with format `hh:mm:ss`, military time. So 14:20:13 is 13 seconds after 2:20pm. The command also sets the maximum number of songs that your music library must be able to hold, to `<MaxSongs>`. You may assume that the `INIT` command will appear exactly once, either as a first command in the cin istream, or as the first command in `<filename>` when the user issues `RUN`. Your program should ignore all but the first `INIT` command, and should reject attempts to add songs to a full library (see the `ADD` command below). *For this command, and all other commands, you may assume well-formed input; that is, you need not perform error checking on the input.*

`ADD <Song Title>, <Artist>, <Runtime>`

Adds the song called `<Song Title>` by the artist or band named `<Artist>` to the music library stored in your program's memory. (Your program should always begin with an empty music library and build it up by responding to `ADD` commands.) The `<Runtime>` field indicates the length of the song, and will take the form 4:32, for example, indicating that it runs for 4 minutes and 32 seconds. Notice that the `<Song Title>`, `<Artist>`, and `<Runtime>` fields are separated by commas.

`PLAY <N>`

Selects and plays `<N>` songs in succession, where N is a number between 1 and 1000, inclusive.

---

[1] The real Pandora analyzes songs' musical traits using the Music Genome project; other systems might try to infer what *else* you might like, based on what other similar users also like.

REST <Time>
Rests for an amount of time indicated in the <Time> field, where <Time> is formatted {h}h:mm:ss. For example, 3:21:40 indicates that the program does not play a song for 3 hours, 21 minutes, and 40 seconds.

LIKE {<Song Title>}
Indicates that the user liked <Song Title>, or if no parameter is included, that she liked the most recent song that played. (See the Implementation section below for what to do with this information.)

DISLIKE {<Song Title>}
Indicates that the user disliked <Song Title>, or if no parameter is included, that she disliked the most recent song that played. (See the Implementation section below for what to do with this information.)

QUIT
Ends the program.

~~

Therefore, the following sequence of commands would initialize the current time to Noon, set the maximum number of songs to 10,000, add 5 songs to the music library, play songs for a while (with some feedback), rest for a little over an hour, add 2 more songs, play some more songs without feedback, and then quit.

```
INIT 12:00:00 10000
ADD Stayin' Alive, The Bee Gees, 3:29
ADD In Your Eyes, Peter Gabriel, 5:27
ADD Hey Ladies, Beastie Boys, 3:48
ADD Perfect Circle, R.E.M., 3:35
ADD We're Going to be Friends, The White Stripes, 2:27
PLAY 1
LIKE
PLAY 1
LIKE
LIKE Stayin' Alive
PLAY 6
LIKE Perfect Circle
PLAY 3
DISLIKE
DISLIKE Human Nature
PLAY 2
REST 1:00:05
ADD This Must be the Place, Talking Heads, 4:56
ADD Human Nature, Michael Jackson, 4:17
PLAY 10
QUIT
```

2. OUTPUT
Your program should simulate the playing of songs by printing the name of the song being played, along with a timestamp. Therefore, if the user requests 4 songs, with a period of 12 hours rest in between, the output might look like the following:

```
[Day 1: 2:45:02pm] Hey Ladies by Beastie Boys
[Day 1: 2:48:50pm] In Your Eyes by Peter Gabriel
[Day 2: 2:54:17am] We're Going to be Friends by The White Stripes
[Day 2: 2:56:44am] Human Nature by Michael Jackson
```

Your output must be formatted exactly as shown. The "Day N" part allows simulated time to cross midnight.

3. IMPLEMENTATION

Store songs in a Max Binary Heap inside a C++ array (you may *not* use `vector` or any other STL container), where the priority of each song (that is, the key value for the purposes of organizing it within the heap) is defined by the following simple formula:

```
(Seconds since this song was last played) + (100 * Likeability)
```

Start every song with a "likeability" factor of 0, and then adjust it according to user feedback, as follows:

- If a user likes a song (using the LIKE command) whose likeability had been 0 or more, then add one to the likeability factor.
- Similarly, if a user uses the DISLIKE command to indicate that she dislikes a song whose likeability was 0 or less, subtract 1 from the likeability factor.
- If a user uses LIKE on a song whose likeability factor was negative, set it to 1. If she DISLIKEs a song whose likeability factor was positive, set it to -1. (This reflects the user "changing her mind" on a song.)

You must support every command in no more than **O(lg N)** expected runtime. Therefore, you must find songs in the Heap more efficiently than by (potentially) searching through the entire heap. To do so, you can overlay a Hash Table and have each entry in that Table store the index in the Heap where the corresponding song is stored. Liking and disliking songs cause the Heap to be reorganized, as does playing a song. Make sure you update both data structures appropriately across all operations.

I am intentionally providing less information than usual about how to complete this assignment. Part of your job is to design an efficient and effective combined data structure.

4. LAB 7

For Lab 7, design, implement, and test a `Time` class (or classes), and a Song class. Demonstrate that your program can parse the input by:

- implementing INIT, RUN, and QUIT completely and correctly
- implementing a placeholder version of ADD that puts songs at the back of an array. (For Lab 7, the array need not be organized as a Heap.)
- accepting PLAY, LIKE, and DISLIKE commands, but not doing anything with them (just output a statement saying that the program received and recognized the command, and echo any parameters).

Think about how you need to use time in your program, and design your `Time` class accordingly. 10% of your grade will reflect the quality of your `Time` class design, and 10% will reflect the quality and thoroughness of your own test program(s) for both the `Time` and `Song` classes.