CS 130: Algorithms and Data Structures I
Fall 2019
Professor Reeves
Assignment 2: MyTunes I
Released: 09/30/2019
Due: 10/11/2019, 11:59 PM

1 Introduction

Digital music services are extremely popular. The popularity of this format started with Apple's release of iTunes which provided consumers with the ability to buy, store, and manage their music on their computer and portable digital music devices such as the iPod and iPhone. Streaming services are very popular today as well. Services like Spotify, Apple Music, and Tidal provide consumers with the ability to listen to virtually any song that they want, when they want. However, you are not satisfied with the digital music products that are out there. And you have decided that you are going to use your CS 130 knowledge to build your own digital music software. This product will be called MyTunes.

The first component that you will build for MyTunes will be the song queue. A queue is implemented to support first in, first out (FIFO) manipulation of the data which it contains. A queue can be implemented in a number of ways. One such implementation is to use an array. However, this array needs to be set up in a specific way to support the operations of a queue. This data structure can be viewed as a "circular" array. By implementing an array in this way, many of the queue operations can be performed in $O(1)$ time. A circular array can be thought of as having $A[1]$ follow $A[0]$, $A[2]$ follow $A[1]$, and so on, up to $A[n-1]$ following $A[n-2]$, but $A[0]$ also follows $A[n-1]$ (which is why the array is considered "circular". The queue is represented by two integers *front* and *rear* that indicate the front and rear elements of the queue, respectively.

2 Instructions

Given the QUEUE data structure below:

```
struct QUEUE {
    string elements[MAX_SIZE];
    int front;
    int rear;
};
```

for this assignment, you are responsible for implementing the following functions for the QUEUE data structure (20 points).

Initialize - This function accepts a previously allocated QUEUE* as a function parameter and initializes the queue to be empty by setting the queue's *front*

index member to 0 and the queue's *rear* index member to -1.

Delete - This function accepts a previously allocated QUEUE* as a function parameter and deletes the memory for the queue and assigns NULL to the QUEUE* parameter.

IsEmpty - This function accepts a QUEUE* parameter as a function parameter and returns *true* if the queue is empty and *false* otherwise. A queue is empty when the *front* index of the queue follows the *rear* index of the queue in a circular sense. For example, if *front* is set to 23 and *rear* is set to 22, the queue is empty. Likewise, if *front* is set to 0 and *rear* is set to MAX_SIZE - 1, the queue is also empty because 0 follows MAX_SIZE - 1 in the circular sense.

IsFull - This function accepts a QUEUE* function parameter and returns *true* if the queue is full and *false* otherwise. A queue is full if it contains MAX_SIZE - 1 elements.

GetLength - This function accepts a QUEUE* function parameter and returns the number of elements currently contained in the queue.

AddElement - This function accepts a QUEUE* function parameter and an *string* function parameter *new_element*. *new_element* is added to the end of the queue if it is not full. The function returns *true* if *new_element* successfully added and *false* otherwise.

AddElements - This function accepts a QUEUE*, an array of *strings* (*elements*), and the length of the array of strings (*array_count*) as function parameters. The elements are added to the end of the queue. If the queue does not have the capacity to hold all of the *new_elements*, elements are added to the queue until it is full. The function returns the number of elements from the *new_elements* array that were added to the queue. Therefore, if *new_elements* contains m elements and all elements are added to the array, m is returned by this function.

FindElement - This function accepts a QUEUE* and a *string*, *element*, as function parameters. It returns the index of *element* if *element* is in the queue and returns -1 if *element* is not in the queue. This function must implement an algorithm that has an asymptotic runtime of $O(\log n)$.

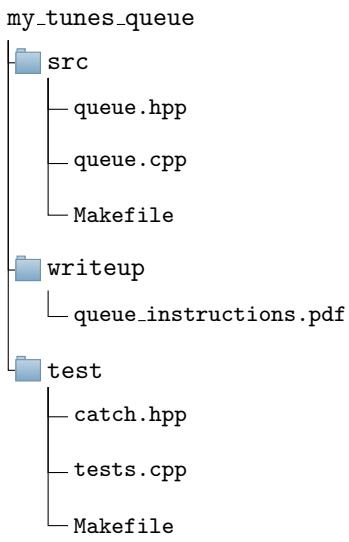
RemoveElement - This function accepts a QUEUE* and an *int*, *r_idx*, as function parameters. If *r_idx* references a valid index in the queue, the element located at *r_idx* is removed from the queue, any elements in the queue after the removed element are moved 1 index closer to the front of the queue, the *rear* index is decreased by 1, and the function returns *true*. If *element* is not in the queue, the function returns *false* and no changes are made to the queue.

SwapElements - This function accepts a QUEUE*, an *int* (*first*), and an *int* (*second*) as function parameters. If both indices are between *front* and *rear*, the position of the elements are swapped so that after this function executes the *element* previously at *second* will be located at *first* and the *element* previously at *first* will be located at *second*. If the swap is successfully performed, the

function returns *true*, otherwise, the function returns *false*.

3 Files

You are being provided with a set of files for this assignment that are available on Blackboard. The files are located in a ZIP file named *my_tunes_queue.zip*. The directory structure provided when you uncompress *my_tunes_queue.zip* is as follows:



Starter files for your implementation have been provided for you. These files are *queue.hpp* and *queue.cpp*. Implement your solution in *queue.cpp*. The *test* directory stores files for testing that your code works as expected. You should make sure that all tests are passed before you submit your assignment. More details on testing are located in the *Testing* section of this document.

4 Testing

Your implementation is being tested using a unit-testing framework called Catch2. The functionality of this library is provided in the file *test/catch.hpp*. Feel free to view this file but **do not** modify it. A Makefile has been provided for you in the *test* directory. To build the tests, type *make* inside the *test* directory from your computer's command line. To run, the tests after they have been successfully built, type *./runtests* in the *test* directory from your computers command line. You can build and run your tests with a single line by typing *make && ./runtests*. When you initially run your tests (before implementing your solution), you will see the following at the end of the output from the *./runtests* command:

```
===============================================================================
test cases: 10 | 2 passed | 8 failed
assertions: 11 | 3 passed | 8 failed
```

After successfully implementing your solutions, you will see a message such as the following in the console after executing the *./runtests* command:

```
g++ -std=c++11 my_tunes_queue.o tests.cpp -I ../src -o runtests
===============================================================================
All tests passed (18 assertions in 10 test cases)
```

You may need to install the GNU g++ and GNU make on your computers in order to properly compile your code and run the tests. Instructions for downloading and installing these programs are included below for Windows:

4.1 g++

`http://www.codebind.com/cprogramming/install-mingw-windows-10-gcc/`

4.2 Make

`http://gnuwin32.sourceforge.net/packages/make.htm` (Windows)

Things are a bit simpler for Mac OS X users. You need to install Xcode and the command line tools (both are free). Instructions can be found here:

`https://www.embarcadero.com/starthere/xe5/mobdevsetup/ios/en/installing_the_commandline_tools.html`

If you are having trouble setting up these programs, please come talk to me about it as soon as possible.

5 Evaluation

The only file that you need to submit is your version of *queue.cpp* on Blackboard for this assignment. If your code passes all of the tests for this assignment, you will receive most (18 points) of the points available. The last 2 points for the assignment will only be awarded if your *FindElement* function is implemented using a $O(\log n)$ algorithm.

6 Bonus

For bonus points on this assignment, explain what advantages and disadvantages exist with this implementation of a queue compared to a queue data structure implemented using a linked list. You can include this explanation in comments at the top of the *queue.hpp* file that you submit.