CX 4010 / CSE 6010
Assignment 2
Software Modules
small 8/28/20 updates highlighted in blue

Due Date: 11:59pm on Thursday, September 3
Submit a single zipfile as described herein to Canvas!

This assignment consists of writing a program to implement a program module for a queue using a linear linked-list data structure and to write a unit `test` to verify the program works correctly. The queue code should be structured as if it were part of a software library to be used with other programs written by someone else. You should assume that the data values are of type `double`. The software should implement the following interface.

1. Use a `typedef` to define a data structure called `LLQueue`, defined as a C `struct` representing the linked-list queue. The implementation of the queue data structure is not visible outside the module you are creating.

2. Write a function with the prototype

   ```
   LLQueue *LLQ_create();
   ```

   Create a new queue and return a pointer to it. The queue is initially empty—that is, it contains no elements. Return `NULL` if the operation fails, e.g., due to a lack of memory.

3. Write a function with the prototype

   ```
   int LLQ_insert(LLQueue *LLQ, double data);
   ```

   Insert the item of type `double` stored in `data` into the queue `LLQ`. Return 0 if the operation succeeds, or an arbitrary value not equal to 0 if it fails.

4. Write a function with the prototype

   ```
   double LLQ_delete(LLQueue *LLQ);
   ```

   Remove the oldest data item from the queue `LLQ`. The function returns the data item that was removed or `0` if the queue is empty; you may assume it is the user's responsibility to not call delete on an empty queue (because size can be checked using `LLQ_count` described below).

5. Write a function with the prototype

   ```
   double *LLQ_Search(LLQueue *LLQ, double data);
   ```

   Search for the value `data` within the queue. The function returns a pointer to the item `data` or `NULL` if the data item is not present.

6. Write a function with the prototype

   ```
   double LLQ_minimum(LLQueue *LLQ);
   ```

   Return the minimum of the items in the queue.

7. Write a function with the prototype

   ```
   double LLQ_maximum(LLQueue *LLQ);
   ```

   Return the maximum of the items in the queue.

8. Write a function with the prototype

   ```
   unsigned int LLQ_count(LLQueue *LLQ);
   ```

   Return the number of items currently in the queue.

9. Write a function with the prototype

   ```
   void LLQ_print(LLQueue *LLQ);
   ```

   Print the data items stored in the queue in order from oldest to newest without changing the contents of the queue.

10. Write a function with the prototype

    ```
    void LLQ_free(LLQueue *LLQ);
    ```

    Delete the queue `LLQ` by releasing all memory used by all the contents of the data structure.

Write a "unit test" program in a separate file from the linked-list queue module to test your queue for a variety of test cases. The program should exercise the linked-list queue using a variety of test cases, including the following.

- Creating the queue.
- Inserting an item into an empty queue.
- Inserting an item into a queue containing at least 4 elements.
- Returning the maximum and minimum of a queue containing at least 4 elements.
- Searching for an item present at the beginning, middle, and end of the linked list for a queue containing at least 4 elements.
- Searching for an item that is not present in a queue containing at least 4 elements (error case).
- Deleting all of the items in a queue containing at least 4 elements, one after the other, finally deleting the queue.
- Deleting an item from an empty queue (error case).

For each test case, use the `LLQ_count()` and `LLQ_print()` functions to verify the operations work correctly.

Your code should include a .h file that defines the interface to the software that includes only the information necessary to use the module. It must *not* specify the internal implementation of the code.

You should submit to Canvas a single zipfile that is named according to your Georgia Tech login—the part that precedes @gatech.edu in your GT email address. To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

The zipfile should include the following files:

(1) your code (all .c and .h files). If you are using linux or Mac OS, we recommend you use a makefile to compile and run your program, and you should include it if so. A sample makefile is provided.

(2) a README text file (not formatted in a word processor, for example) that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program.

(3) a series of 4 slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named slides.pptx or slides.pdf, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). In other words, under the assumption that the mathematics behind what you are doing is already known, what were the main things you did to translate those requirements into code? You are limited to one slide.
- Slides 2 and 3: a description of your most important tests and sample output produced by the test program. You are limited to two slides.
- Slide 4: a brief exposition of what you felt were the most challenging one or two aspects of this assignment, limited to one slide.

Some hints:

- Start early!
- Identify a logical sequence for implementing the modules. Then implement one at a time, and verify each works properly before proceeding to implement the next module.