Programming Assignment #3

This assignment will help you get acquainted with the basics of queues, malloc, free, pointers, and structs. This is the third assignment that will be for a grade. Please take the time to read through the instructions and complete the function below.

Setup

Begin by downloading the starter files provided on Canvas and creating a new project on VS Code.

UT Queue

In this project, we are creating an Abstract Data Type (ADT) to implement our own queue. We will use malloc and free, and get some practice with pointers, structs, and macros.

A queue is a data structure that follows the FIFO (first-in, first-out) principle for adding and removing elements. Our UT Queue will deal with integers (**int**) as the datatype for the contents of the queue. For more information, visit this link.

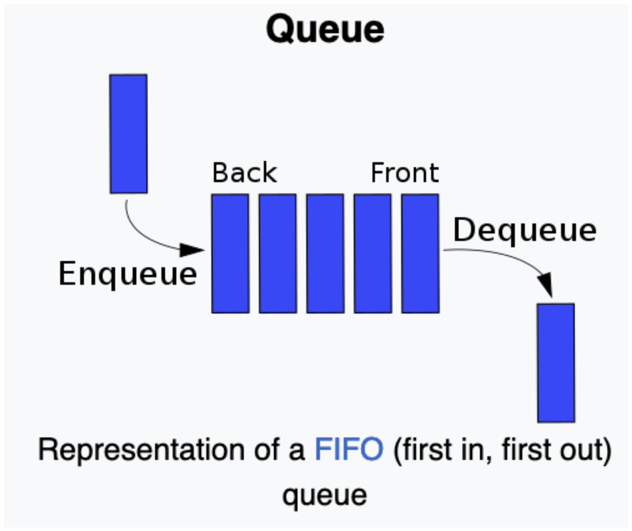

Figure 1: Image representation of the FIFO queue.

It is your job to implement UT Queue ADT. You must complete the assignment **without modifying** the `UTQueue` struct that is defined inside UTQueue.h. The struct consists of the following parts:

- `size`: the number of elements in the queue.
- `data`: a pointer to an array that must be dynamically allocated.

Example of a `UTString`:

`size` = 5
`data` = [1, 2, 3, 4, 5]

Functions

For this lab, you will implement the following functions:

- `UTQueue* utqueuedup(const int* src, int n)`: Create a UTQueue on the heap that holds a copy of source (which has n items) and sets the size. Return a pointer to the UTQueue.
- `void utqueuefree(UTQueue *self)`: frees **self**. Must deallocate both **data** and the **UTQueue** itself.
- `UTQueue* utqueuepush(UTQueue *src, int value)`: **Push** (aka **enqueue**) the integer **value**. This will put the new integer at the end of the queue and increase the size by 1. Keep in mind that you will have to resize **data** for the new value to fit. Return the pointer to the modified UTQueue.
- `UTQueue* utqueuepop(UTQueue *src)`: **Pop** (aka **dequeue**) an element from the queue. This will cause the size to decrease by 1. This function should fail an **assert()** statement if the src UTQueue is already empty and the program should immediately crash this can be done using **assert(src⟶size != 0)**. It is recommended to resize **data** (make it smaller) to avoid having garbage values in data. Return the value of the integer popped from the UTQueue.
- `UTQueue* utqueuerev(UTQueue *src)`: Reverses the contents of the UTQueue (what is in **data**). Return the pointer to the reversed UTQueue.
- `UTQueue* utqueuecomb(UTQueue *dst, UTQueue *src)`: Combines two queues by adding the data from **src** to the data of **dst**. **AFTER COPYING THE VALUES FROM SRC, FREE SRC**. Return the pointer to **dst**.
- `UTQueue* utqueueswap(UTQueue *q1, UTQueue *q2)`: Swap the two queues (i.e., the data and size from q1 should be moved to q2 and vice versa. There are many ways to go about doing this. No need to return anything from this function.
- `UTQueue* utqueuecopy(UTQueue *dst, const int *src, int n)`: Copy as many elements from the integer array **src** into the UTQueue **dst**. Overwrite values of the Queues data. **n** is the number of elements in the src array. For example, if src has five elements and dst only has a size of three, copy the first three elements from src into dst. If src only has three elements and dst has a size of 10, copy all 3 elements from src into the first three locations in the queue - leave the rest of the queue alone. Return the pointer to the modified UTQueue.

Bounds and Requirements

size must always be equal to the number of elements in **data**.

UTQueues must be stored on the heap. UTQueues will **ONLY** be created by calling `utqueuedup`.

PLEASE use Valgrind to make sure your code isn't causing any memory leaks. This could significantly affect your grade. To do this, copy your files to a folder in your ecelrc (Linux server) home directory, compile with g++ *.cpp, run the executable to make sure it works with the command ./a.out, and then finally use Valgrind to check for memory leaks by using the command valgrind ./a.out.

Submission

Submit only your Project3.cpp file to Gradescope. **Please do not edit the name of this file**; the grader will give you a 0. After a short time, a few test case results should be visible to you on Gradescope. These are sample test cases, letting you know your Gradescope submission compiled and ran successfully on those tests. **Please note passing the sample cases DOES NOT MEAN you scored a 100 on the assignment.** Debug, debug, debug, and test, test, test!

Please do your own work on the project and do not share your code with others.

Good luck and have fun!

Workshop

A reminder that we host an assignment workshop (extra office hour) the Friday before each assignment is due. If you ever need help with your assignment, feel free to stop by! Your TAs are always willing to help and we want you to succeed in the class!