**CMPS 12M-02**
**Data Structures Lab**
**Lab Assignment 5 (Lab5)**
**Due: Saturday, December 2 by 11:59pm**

- **Assignments submitted late will lose 1 point if submitted less than one day (24 hours) late.**
- **Assignments submitted more than one day late will receive no credit.**

The goal of this assignment is to learn how to implement ADTs in C, even though it is not an object-oriented language. We discuss the *typedef* and *struct* commands, header files, information hiding, constructors and destructors, and memory management in a separate file called Some_Additional_Background_on_C.pdf. You will write a program in C that creates a simple Queue ADT. Please read the file Some_Additional_Background_on_C.pdf first, before reading this file.

**What to Turn In**

Implement a Queue of ints in C. This should support the obvious operations of Enqueue and Dequeue. We also require one additional function, Print, that prints the current queue in order. You should implement the queue as a linked list (Reference-Based implementation). Do not assume that the list has limited fixed size, so don't use an Array-Based implementation. And as usual, you should not use libraries for your linked lists or for queues. You may use a single external reference (as in Lab4) if you'd like, but you may also use two external references if you prefer.

The interface for the Queue ADT should be in the file `queue.h` that you must write. This file contains the descriptions of the above functions. The actual code for them will be in `queue.c` that you also must write.

Finally, write a "client" called `queueClient.c`, which is compiled into the executable `queueClient` This client contains the main function that will read from an input file, and perform all the operations, including printing to an output file. Your client will take two command line arguments naming the input and output files respectively, just as in Lab4.

The input file consists of a series of queue-related instructions. For example, the input file could be:

```
E 45
E 42
E 21
P
D
P
D
P
D
P
D
E 32
```

This means, "Enqueue 45, Enqueue 42, Enqueue 21, Print, Dequeue, Print, Dequeue, Print, Dequeue, Print, Dequeue, Enqueue"

For this input file, the output file must contain:

```
Enqueue 45
Enqueue 42
Enqueue 21
Queue is 45 42 21
Dequeue 45
Queue is 42 21
Dequeue 42
Queue is 21
Dequeue 21
Queue is empty
Dequeue fails
Enqueue 32
```

When your program is complete, test it on various input files, including the one given above. You are <u>not given</u> any other test input or output file for Lab5.. Also, you must check your program for memory leaks by using the unix program `valgrind`. Do:

```
% valgrind --leak-check=full queueClient infile outfile
```

to run `valgrind` on your program with input file `infile` and output file `outfile`. `valgrind` does not work correctly on all versions of Mac OSX (or on Windows), so we recommend that you do this on the Unix timeshare, which is also where we'll test your solution. Do:

```
% valgrind –help
```

to see some of the options to `valgrind`, and see the `valgrind` man pages for further details.

You must also write a Makefile for Lab5 that creates an executable binary file called `queueClient` and includes a `make clean` to delete old binaries. Also include a `make check` in your Makefile which runs `valgrind` on your executable as shown above to check for memory leaks. You may use the Makefile from Lab4 as a starting point, since Lab4 also used an input file and an output file.

If you want to use booleans (not required), remember to include stdbool.h, which provides macros for booleans, which are not natively part of C.

Your Lab5 directory should contain:

```
README
Makefile
queue.c
queue.h
queueClient.c
```

As always start early and ask for help if anything is not completely clear.

**Grading (multiple errors will lead to multiple deductions):**

You code should terminate within 3 minutes for all runs. (Our test data won't be much longer than the example shown above.) If it doesn't, we will not give you credit. There is no checker requirement for this assignment. However, to receive credit, you must also submit your git commit id using the Google Form for Lab5. You can resubmit the Google Form if you want to do so, but we'll grade you based on your latest re-submission, so be careful about this. At this point, everybody should be able to do this correctly!

- (10 points) Everything is done correctly
- (8 points) Code is correct, but Makefile is incorrect
- (8 points) Everything is correct, but valgrind reports a memory leak. queueClient does not crash.
- (6 points) queueClient makes occasional mistakes but does not crash.