CSC190: Computer Algorithms and Data Structures
Lab 5
Assigned: Feb 23, 2015; Due: Mar 2, 2015 @ 10:00 a.m.

1 Objectives

Implementations in this lab will serve as helper functions for Assignment 2. You will implement interfaces to create and operate on a queue data structure. We have provided files that you must use as a starting point to complete this lab. You will download contents in the folder `Lab5` which contains two folders (`code` and `expOutput`) into your ECF workspace. Folder `code` contains a skeleton of function implementations and declarations. Your task is to expand these functions appropriately in `queue.c` and include necessary libraries in `queue.h` as required for implementation. `main.c` evokes all the functions you will have implemented and is similar to the file that will be used to test your implementations. Use `main.c` file to test all your implementations. Folder `expOutput` contains outputs expected for the supplied `main.c` file. Note that we will **NOT** use the same `main.c` file for grading your lab. Do **NOT** change the names of these files or functions.

2 Grading

It is **IMPORTANT** that you follow all instructions provided in this lab very closely. Otherwise, you will lose a *significant* amount of marks as this lab is auto-marked and relies heavily on you accurately following the provided instructions. Following is the mark composition for this lab (total of 20 points):

- Successful compilation of all program files with no memory leaks i.e. the following command results in no errors (2 points):
  ```
  gcc main.c queue.c -o run
  valgrind --quiet --leak-check=full --track-origins=yes ./run 1
  ```
- Output exactly matches expected output (10 points)
- Code content (8 points)

Sample expected outputs are provided in folder `expOutput`. We will test your program with a set of completely different data files. Late submissions will **NOT** be accepted.

Part 1: Defining Interfaces and Structures of a Queue (Linked)

In this part, function implementations of `enqueue`, `dequeue` and `freeQueue` will be tested. These functions are to be defined in the `queue.c` file. Prototypes of these functions and structures associated with the queue data structure reside in `queue.h`. The underlying implementation of the queue data structure will be based on linked lists. Three structures to be defined first are:

- `Data`
- `Node`
- `Queue`

The `Data` structure will consist of two `double` members: `arrivalTime` and `departureTime`. The structure `Node` has two members: `data` and `next`. `data` is of type `struct Data` and `next` is a pointer of type `struct Node`. The third structure to be defined is `Queue` which has three member variables: `currSize`, `front` and `rear`. `currSize` stores the number of nodes currently residing in the queue. `front` points to the first node in the queue and `rear` points to the last node in the queue. You will implement the following four functions that will operate on the queue data structure:

- `struct Queue initQueue();`

- `void enqueue(struct Queue *qPtr, struct Data d);`
- `struct Data dequeue(struct Queue *qPtr);`
- `void freeQueue(struct Queue *qPtr);`

The function `initQueue` will initialize a queue structure by setting the members `currSize` to 0 and pointer members `front` and `rear` to NULL. The `enqueue` function will insert the node `d` into the back of the queue represented by the pointer `qPtr`. The `dequeue` function will remove a node from the front of the queue represented by the pointer `qPtr` and return the data contained in this node. `freeQueue` will use the `dequeue` function to free all nodes in the queue pointed to by `qPtr`. This part of the assignment will be tested via the following commands:

- `./run 1`
- `valgrind --quiet --leak-check=full --track-origins=yes ./run 1`

Outputs from these tests must match content of `P1.txt` which is the result of the parameters passed from function calls in `main.c`.

3 Code Submission

For this lab, if your submit via `git`, you will receive **bonus** points. Otherwise you can submit through the `submitcsc190s` command on your ECF machine (no bonus points). Ensure that you submit through only one venue.

3.1 Submission through `Git`

Once you have completed this lab, you will submit your work by:

- Log onto your ECF account
- Browse into the directory you had cloned in Lab 0 (i.e. `cd ~/UTORID/`)
- Create a folder named `Lab5` (i.e. `mkdir Lab5`) in that cloned directory
- Ensure that your code compiles in the ECF environment
- Copy all your completed code (`queue.h` and `queue.c`) into the Lab5 folder
- Browse into the `~/UTORID/` directory
- Add all files in the Lab5 folder (i.e. `git add *`)
- Commit all files that have been modified in the Lab 5 folder (i.e. `git commit -m "adding lab files"`)
- Push all changes committed to the `git` server (i.e. `git push origin master`)

3.2 Submission through `submitcsc190s`

- Log onto your ECF account
- Ensure that your completed code compiles
- Browse into the directory containing your completed code (`queue.h` and `queue.c`)
- Submit by issuing the command:
  `submitcsc190s 6 queue.h queue.c`

3.3 Checklist

ENSURE that your work satisfies the following checklist:

- You submit before the deadline
- All files and functions retain the same original names
- Your code compiles without error in the ECF environment (if it does not compile then your maximum grade will be 3/20)
- Do not resubmit any files in Lab 5 after the deadline (otherwise we will consider your work to be a late submission)