CSC190: Computer Algorithms and Data Structures
Lab 6
Assigned: Mar 16, 2015; Due: Mar 23, 2015 @ 10:00 a.m.

1 Objectives

The purpose of this lab is to use queue and heap ADTs to build a **priority queue**. Priority queue is an example of an ADT that is built upon other ADTs. In the lectures, we have seen an implementation of a *max heap*. In a max heap, parent nodes have larger values than descendants. In this lab, you will implement a *min heap*. In a min heap, parent nodes have elements that are smaller than descendants. Like a max heap, min heap is always complete. When an element is deleted from the heap, the smallest element in the tree is removed and returned by the function that is performing the removal. If all elements are deleted from the heap and inserted into a queue, then all elements in the queue will be ordered from lowest (at the front) to the highest (at the back). This queue is called a *priority queue*. You will implement this in the lab.

You will download contents in the folder `Lab6` which contains two sub-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 `circQueue.c` and `heaps.c` 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 circQueue.c heaps.c main.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.

Heap Implementation

First, you will define all interface functions associated with the `minHeap` listed in the file `heaps.c`:

- `struct Heap * initMinHeap()`
- `void insertMinHeap(struct Heap * h, struct Data d)`
- `struct Data removeMinHeap(struct Heap * h)`
- `void freeMinHeap(struct Heap * h)`

All structures are defined in the `lab6.h` file. As you may notice, the `Heap` structure uses a sequential representation to store the heap (i.e. via an array). Hence all your interface functions should be based on a sequential implementation. `initMinHeap` function initializes all members of a dynamically created `struct Heap` variable and return a pointer to this variable. The `insertMinHeap` function inserts a new node of type `struct Node` containing the data `d` passed to the function into the heap pointed to by `h`.

This insertion should be similar to the heap insertion covered in class. The new node is placed at the next available level-order spot in the bottom-most level. Nodes with smaller data values are bubbled up instead of larger nodes (in order to maintain the min heap tree property where every parent node is smaller than its descendants). `removeMinHeap` function deletes the root node and returns the `struct Data` variable stored in the root node of the heap pointed to by `h`. The root node is replaced by the last level-order node in the bottom-most level. Nodes with larger data values are bubbled down. Finally, the `freeMinHeap` function will free the dynamically allocated heap variable pointed to by `h`.

Priority Queue

In a priority queue, data value of nodes increase in value from the `front` to the rear of the queue and thus termed the priority queue. The min heap can be used to store all nodes. Then nodes can be removed from the min heap one by one and enqueued into the queue. This will ensure that all nodes in the queue range from lowest to highest priority. The priority queue has the same interface functions as a regular queue:

- void initQueue(struct Queue ** qPtr);
- int isEmpty(struct Queue * Q);
- void enqueue(struct Queue * Q, struct Data d);
- void dequeue(struct Queue * Q, struct Data *d);
- void freeQueue(struct Queue * Q);

One condition is that this queue should be implemented with a circular array. The definition of the `Queue` structure is provided in the lab6.h file. `initQueue` shall initialize members in a dynamically created `Queue` structure and indirectly return a pointer to this variable via `qPtr`. `isEmpty` returns 1 if there are no nodes in the queue and 0 otherwise. Into the function `enqueue`, `struct Data` variable which has been removed from the min heap is passed as an argument `d` and stored at the rear of the Queue pointed to by `Q`. `dequeue` will remove the node at the front of the Queue pointed to by `Q`. `freeQueue` will free dynamically allocated components.

Testing Implementation

This part of the lab 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 are the results 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 `Lab6` (i.e. `mkdir Lab6`) in that cloned directory

- Ensure that your code compiles in the ECF environment
- Copy all your completed code (`circQueue.c` and `heaps.c`) into the Lab6 folder
- Browse into the `~/UTORID/` directory
- Add all files in the Lab6 folder (i.e. `git add *`)
- Commit all files that have been modified in the Lab 6 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 (`heaps.c` and `circQueue.c`)
- Submit by issuing the command:
  `submitcsc190s 8 circQueue.c heaps.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 6 after the deadline (otherwise we will consider your work to be a late submission)