Programming Assignment 5 - PART-I: A Priority Queue Module

Tentative Due Date: ~~Wed, April 20~~ Friday, April 22 at 10:59AM

In this project you will implement a priority queue module using binary heaps. The interface is specified in the given file pq.h. For this project, you will have to complete the source file pq.c *completely* from scratch.

When a priority queue is created, it's capacity is specified and does not change (i.e., no dynamic resizing).

The header file contains comments for each function describing their semantics and giving runtime requirements. It is reproduced below.

```
/**
* General description:  priority queue which stores pairs
*   <id, priority>.  Top of queue is determined by priority
*   (min or max depending on configuration).
*
*   There can be only one (or zero) entry for a particular id.
*
*   Capacity is fixed on creation.
*
*   IDs are integers in the range [0..N-1] where N is the capacity
*   of the priority queue set on creation.  Any values outside this
*   range are not valid IDs.
**/



// "Opaque type" -- definition of pq_struct hidden in pq.c
typedef struct pq_struct PQ;


/**
* Function: pq_create
* Parameters: capacity - self-explanatory
*             min_heap - if 1 (really non-zero), then it is a min-heap
*                        if 0, then a max-heap
*
```

```
* Returns:  Pointer to empty priority queue with given capacity and
*           min/max behavior as specified.
*
*/
extern PQ * pq_create(int capacity, int min_heap);

/**
* Function: pq_free
* Parameters: PQ * pq
* Returns: --
* Desc: deallocates all memory associated with passed priority
*       queue.
*
*/
extern void pq_free(PQ * pq);

/**
* Function: pq_insert
* Parameters: priority queue pq
*             id of entry to insert
*             priority of entry to insert
* Returns: 1 on success; 0 on failure.
*          fails if id is out of range or
*            there is already an entry for id
*          succeeds otherwise.
*
* Desc: self-explanatory
*
* Runtime:  O(log n)
*
*/
extern int pq_insert(PQ * pq, int id, double priority);

/**
* Function: pq_change_priority
* Parameters: priority queue ptr pq
*             element id
*             new_priority
* Returns: 1 on success; 0 on failure
* Desc: If there is an entry for the given id, its associated
*       priority is changed to new_priority and the data
*       structure is modified accordingly.
*       Otherwise, it is a failure (id not in pq or out of range)
* Runtime:  O(log n)
*
*/
extern int pq_change_priority(PQ * pq, int id, double new_priority);
```

```
/**
* Function: pq_remove_by_id
* Parameters: priority queue pq,
*             element id
* Returns: 1 on success; 0 on failure
* Desc: if there is an entry associated with the given id, it is
*       removed from the priority queue.
*       Otherwise the data structure is unchanged and 0 is returned.
* Runtime:  O(log n)
*
*/
extern int pq_remove_by_id(PQ * pq, int id);

/**
* Function: pq_get_priority
* Parameters: priority queue pq
*             elment id
*             double pointer priority ("out" param)
* Returns: 1 on success; 0 on failure
* Desc: if there is an entry for given id, *priority is assigned
*       the associated priority and 1 is returned.
*       Otherwise 0 is returned and *priority has no meaning.
* Runtime:  O(1)
*
*/
extern int pq_get_priority(PQ * pq, int id, double *priority);

/**
* Function: pq_delete_top
* Parameters: priority queue pq
*             int pointers id and priority ("out" parameters)
* Returns: 1 on success; 0 on failure (empty priority queue)
* Desc: if queue is non-empty the "top" element is deleted and
*       its id and priority are stored in *id and *priority;
*       The "top" element will be either min or max (wrt priority)
*       depending on how the priority queue was configured.
*
*       If queue is empty, 0 is returned.
*
* Runtime:  O(log n)
*
*
*/
extern int pq_delete_top(PQ * pq, int *id, double *priority);

/**
```

```
* Function:  pq_capacity
* Parameters: priority queue pq
* Returns: capacity of priority queue (as set on creation)
* Desc: see returns
*
* Runtime:   O(1)
*
*/
extern int pq_capacity(PQ * pq);

/**
* Function: pq_size
* Parameters: priority queue pq
* Returns: number of elements currently in queue
* Desc: see above
*
* Runtime:  O(1)
*/
extern int pq_size(PQ * pq);

/**
* Function: pq_contains
* Parameters: priority queue pq
*             element id
* Returns: 1 if there is an entry in the queue for given id
*          0 otherwise
* Desc: see above
*
* Runtime:  O(1)
*
* Note:  same return value as pq_get_priority
*
*/
extern int pq_contains(PQ * pq, int id);
```

Readme File:

To make grading more straightforward (and to make you explain how you achieved the assignment goals), you must also submit a Readme file.

The directory containing the source files and this handout also contains a template Readme file which you should complete (it is organized as a sequence of questions for you to answer).

Submission:

You will submit an archive file containing both your source code file pq.c and your readme file.

Note: pq.c should not contain a main function. You will certainly have developed one or more main drivers to test your implementation of the priority queue. But these should be in other "client" files (which you do not have to submit) or, if you wrote a tester/driver in pq.c, just be sure to comment it out or disable it in some way.

Aside: a handy trick in C is to use the preprocessor to enable/disable chunks of code -- like a main function. Here is an example:

```
// file:  util.c
/** lots of functions ***/



/** END OF UTILITY FUNCTIONS **/

#ifdef DEV

// driver program exercising above functions
int main(){

  // blah blah


}

#endif
```

`$ gcc -DDEV util.c`	This will behave the same as if we inserted `#define DEV` At the top of the source file and

	produce an executable.
$ gcc -c util.c	This on the other hand would compile as if the main function was not there at all -- producing a .o file to which other programs can link.
This is kind of nice because you get two behaviors without changing the source code. (You could even have multiple main functions each enabled by a different preprocessor symbol. By the way, there is nothing magic about “DEV” -- I just chose it).	
With respect to this assignment, you would be “safe” if you employed this strategy -- we will be able to compile your code into a .o; link our own driver/test to that .o and run our tests.	