Jacob Shay

My implementation is a priority blocking queue that users can enter information into the queue and it will sort it into the right priority. The priority queue takes elements which it then compares two at a time to determine which one is bigger. My priority queue allocates inserts, finds the max element, finds the max element and removes it, and tells if a queue is empty or not.

A priority queue is an abstract data type which is like a regular queue or stack but where each element has a priority associated with it. In a priority queue an element that has a higher priority level will be dealt with before an element with a lower priority will be dealt with. If both elements have the same priority than they will be dealt with according to the order they were entered in a queue.

References:

* <http://andreinc.net/2011/06/01/implementing-a-generic-priority-queue-in-c/> Retrieved on November 19th, 2013.
* <http://rosettacode.org/wiki/Priority_queue> Retrieved on November 1st, 2013.
* <http://see-programming.blogspot.com/2013/05/c-program-to-implement-priority-queue.html?m=1> Retrieved on November 5th, 2013.

My priority blocking queue takes elements and their priorities and then orders them according to their priority. This would be useful for someone making a program that has tasks and has to order them into a priority that needs to get things done.

Software Features:

* Allocates memory for a given priority queue
* Add an element into the priority queue
* Find the element with the top priority in the queue
* Fine the element with the top priority in the queue and remove it. Then restructure he queue.
* Tells the user if the queue is empty or not
* The ability to destroy and de-allocate memory for a given queue

Users:

* Anyone wanting to order tasks
* Compare elements

Constraints:

* Length of the priority queue has to be established. Cannot go over the size established for the queue.

Assumptions and Dependencies

* My priority queue will work on strings, and integers. It was able to compare strings and return the right order the strings were supposed to be in. Also it was able to count the priorities for each element with integers and be counted accordingly.
* The priority queue should be able to handle any data type, however, I have not tested it, and it should be able to work with anything as long as the compare function is written correctly.
* The priority queue can edited to fit specific users’ needs or ideas.

Design Specifics:

Functions:

* **typedef struct PQueueStruct**-Creates a priority queue with the right data has a size, capacity, the data being entered or removed.
* **Priority\_queue \*pq\_create**-allocates memory for a given priority queue.
* **void pqeue\_destory(priority\_queue \*q)**-destroys the priority queue structure and frees up memory.
* **void pqueue\_heapify(priority\_queue \*q, size\_t idx)**-compares elements till it finds the largest. At this point it swaps data until they are in the right order and then structures the given queue.
* **int pq\_empty(priority\_queue \*q)**-returns 1 if the size of the given queue is empty else it returns 0 if the size of the given queue is greater than 0.
* **void pq\_insert(priority\_queue \*q, const void \*data)**-adds given data to the queue. If the element is not in the right position based on priority it is then swapped until all the elements are in the right position.
* **void \*pq\_max(priority\_queue \*q)**-goes through the queue and returns the largest element in the queue.
* **void \*pq\_Remove\_max(priority\_queue \*q)**-removes the top element from the queue and then reorders the structure based on the remaining elements.
* **Int word\_compare(const void \*d1, const void \*d2)**-compares two string words and determines which one is larger and returns the strings in order.
* **Int count\_compare(const void \*d2, const void \*d2)**-compares priorities that are associated with a certain word. Each word in this example has a count which is based on the order the words go in.
* **Int number\_cmp(const void\*d1, const void \*d2)**-sorts numbers into their correct order.

Files:

* **Final.c**-The C file where the code is written to write the program.
* **Final.exe**-An executable that we can run to test our code using the terminal. (best idea to compile it before trying to run it)

Testing:

* NOTE: For each of these tests I created 3 words of structure type. Each word has an string value, and a count value (for priority in the queue);
* Word1.str=”One”
* Word1.count = 100;
* Word2.str = “Two”
* Word2.count = 50
* Word3.str = “Three”
* Word3.count = 20;

1. I created a priority queue and named it pq1. I was using the **word\_compare** function here to compare the strings for each word. I then inserted the word structures for this priority queue. I then wrote a while statement and said while the queue was not equal to the function **pq\_emtpy(pq1)** then to go through the queue find the largest element based on the string for each word and display it and then remove the max value each time until the queue was empty.

* Results for pq1:
* Two
* Three
* One

1. I then created a priority another priority queue and named it pq2. I was using the **count\_compare** function this time to compare the priority of each word. I inserted the word structures for pq2. I then wrote a while statement and said while the queue was not equal to the function **pq\_empty(pq2)** then to go through the queue and find the largest element based on the count of each word and display it and then remove it until the queue was empty.

* Results
  + One
  + Two
  + Three

1. Then for the third queue that I created was called pq3. It uses the **number\_cmp()** function to compare the numbers to see which one is lower or not. This time I create a for loop starting at 1 and going through till 1 was less than or equal to five. As this goes through the loop it then assigns a number between 1 and 5 for I, to a pointer called number. Then in that same iteration I insert the number into the queue using the **pq\_insert()** function. Then after all five numbers are in i find the max element in the queue using the **pq\_max()** function to find the max value and return it. Then I use the same while loop until the queue is empty to find the max number and remove it.

* Results:
* 5
* 5
* 4
* 3
* 2
* 1

Developers Guide:

Anyone can use this implementation. All they would need to do is have a way to compile and run the c code. I used Ubuntu Linux for this and the version was 13.01 64 bit downloaded directly from there website at: <http://www.ubuntu.com/download>. If anyone would be interested in using my code in the future then in order to compile this than they would just need to use the “gcc Final.c –o Final” to compile it without the quotation marks.