Assignment #3: Priority Queue

CS201 Fall 2019
15 points
due Monday, Sept. 30th, 11:59 pm

1 Priority Queue

A priority queue is a list of elements, maintained in sorted order according to the *priority* of each element. For the purposes of this assignment, the priority will be a nonnegative integer, and the highest-priority element will be the one whose priority value is the smallest. The list will be maintained in ascending priority order, so that the highest-priority element will be at the front of the list.

Use these structures:

```
typedef struct {
   int id;
   char name[32];
} StudentRecord;
```

```
typedef struct PQueueStruct {
   int priority;
   void *data;
   struct PQueueStruct *next;
} PQueueNode;
```

and implement these functions:

```
int enqueue(PQueueNode **pqueue, int priority, void *data)
```

This will put the data in the priority queue in the correct place. If there is already one or more nodes in the list having the same priority as the data that you are enqueueing, then put the new node after all of the nodes having that priority. Return zero from this function.

```
void *dequeue(PQueueNode **pqueue)
```

Remove the front of the list and return the *data* from that list node (not the list node itself). If the pqueue is empty, then return NULL.

```
void *peek(PQueueNode *pqueue)
```

Return the *data* from the first node in the pqueue (not the node itself). If the pqueue is empty, then return NULL. The peek operation does not actually remove a node from the pqueue!

```
void printQueue(PQueueNode *pqueue, void (printFunction)(void*))
```

Print the data from each node in the queue.

```
int getMinPriority(PQueueNode *pqueue)
```

Return the priority of the first node in the pqueue. If the pqueue is empty, return -1.

```
int queueLength(PQueueNode *pqueue)
```

Return the number of nodes in the pqueue.

```
void printStudentRecord(void *data)
```

Print an instance of StudentRecord as `"%s %d"`, with the name field and id field.

Each priority-queue node will be a container for an instance of some other data structure—this keeps the definition of the queue node independent of the specific data that the queue node contains. The `void*` pointer can point to anything—it's up to the user of the priority queue to cast the pointer to a pointer of the appropriate underlying data type.

2 Passing a function as a parameter

To keep this independence between the queue nodes and the underlying data (and since we're not using an object-oriented language), when we want to print the contents of a queue node (specifically, the object to which the queue node is pointing), it's necessary to pass in a function that specifies how to print the underlying data object. For an example of how to use a function as a parameter, see function-parameter.c on the course gitlab site (https://gitlab.uvm.edu/Jason.Hibbeler/ForStudents/blob/master/CS201/Assignments/three/function-parameter.c)

3 What to Submit

Submit only two files: pqueue.netid.h and pqueue.netid.c. Do not put a main() in your file. Put only the seven functions described above.

4 Testing Your Code

- Create a student named John, with id = 67. Enqueue this student with priority = 3.
- Create a student named Brittany, with id = 890. Enqueue this student with priority = 8.
- Create a student named Robert, with id = 645. Enqueue this student with priority = 1.
- Create a student named Alice, with id = 112. Enqueue this student with priority = 9.
- Create a student named Thomasina, with id = 452. Enqueue this student with priority = 2.
- Create a student named Alfred, with id = 516. Enqueue this student with priority = 12.
- Create a student named Margaret, with id = 341. Enqueue this student with prioirity = 9.

- Create a student named Robert, with id = 485. Enqueue this student with priority = 2.
- Create a student named Elizabeth, with id = 734. Enqueue this student with priority = 1.

Then print your queue. You should see this:

tt priority = 1 data = Robert 645
priority = 1 data = Elizabeth 734
priority = 2 data = Thomasina 452
priority = 2 data = Robert 485
priority = 3 data = John 67
priority = 8 data = Brittany 890
priority = 9 data = Alice 112
priority = 9 data = Margaret 341
priority = 12 data = Alfred 516

Now, do the following:

- peek: should get Robert 645
- dequeue from the queue and print the student record: should be Robert 645
- peek: should get Elizabeth 734
- dequeue from the queue and print the student record: should be Elizabeth 734
- peek: should get Thomasina 452
- create a student David, with id = 908. Enqueue this student with priority = 1.
- create a student Katherine, with id = 267. Enqueue this student with priority = 2.
- create a student Andrew, with id = 372. Enqueue this student with priority = 20.

Then print the queue. You should see this:

tt priority = 1 data = David 908
priority = 2 data = Thomasina 452
priority = 2 data = Robert 485
priority = 2 data = Katherine 267
priority = 3 data = John 67
priority = 8 data = Brittany 890
priority = 9 data = Alice 112
priority = 9 data = Margaret 341
priority = 12 data = Alfred 516
priority = 20 data = Andrew 372

Finally, do this:

```
// studentRec is a variable of type StudentRecord*
// minPriority is a variable of type int

while (queueLength(pqueue) > 0) {
```

```
    minPriority = getMinPriority(pqueue);
    printf("min priority = \%d\n", minPriority);
    studentRec = dequeue(\&pqueue);
    printf("dequeued: ");
    printStudentRecord(studentRec);
}
```

and you should see this output:

```
min priority = 1
dequeued:  David 908
min priority = 2
dequeued:  Thomasina 452
min priority = 2
dequeued:  Robert 485
min priority = 2
dequeued:  Katherine 267
min priority = 3
dequeued:  John 67
min priority = 8
dequeued:  Brittany 890
min priority = 9
dequeued:  Alice 112
min priority = 9
dequeued:  Margaret 341
min priority = 12
dequeued:  Alfred 516
min priority = 20
dequeued:  Andrew 372
```