```
1 /*****************************************************************************
2  File name:  queue.h
3  Author:     $Author: chadd $
4  Date:       Oct 7, 2011
5  Class:
6  Assignment:
7  Purpose:
8  RevisionID: $Id: queue.h 82 2011-10-07 22:15:21Z chadd $
9  *****************************************************************************/
10
11 #ifndef QUEUE_H_
12 #define QUEUE_H_
13
14 #include "../../CS300StaticPriorityQueue/include/pQueue.h"
15
16 typedef short int Q_ERRORCODE;
17
18 // BOOLEAN is provided by list.h
19
20 // Queue error codes for each function to use
21 #define Q_NO_ERROR              0
22
23 // Queue create failed
24 #define Q_ERROR_NO_QUEUE_CREATE   -1
25
26 // user tried to operate on an empty Queue
27 #define Q_ERROR_EMPTY_QUEUE       -2
28
29 // user tried to add data to a full Queue
30 #define Q_ERROR_FULL_QUEUE        -3
31
32 // user tried to operate on an invalid Queue. An invalid
33 // Queue may be a NULL QueuePtr or contain an invalid List
34 #define Q_ERROR_INVALID_QUEUE    -9
35
36 // user provided a NULL pointer to the function (other than the QueuePtr)
37 #define Q_ERROR_NULL_PTR         -10
38
39 typedef struct Queue
40 {
41   PriorityQueue sTheQueue;
42 } Queue;
43
44 typedef Queue * QueuePtr;
45
46 /*****************************************************************************
47 *                    Allocation and Deallocation
48 *****************************************************************************/
49 Q_ERRORCODE queueCreate (QueuePtr);
50 // results: If Queue can be created, then Queue exists and
51 // is empty returning Q_NO_ERROR; otherwise,
52 // Q_ERROR_NO_QUEUE_CREATE is returned
53
54
55 Q_ERRORCODE queueTerminate (QueuePtr);
56 // results: Queue no longer exists
57 // Q_ERROR_INVALID_QUEUE
58
59 /*****************************************************************************
60 *                 Checking number of elements in queue
61 *****************************************************************************/
62 Q_ERRORCODE queueSize (Queue, int *);
63 // results: Returns the number of elements in the Queue
64 // Possible errors:
```

```
65 // Q_ERROR_INVALID_QUEUE
66 // Q_ERROR_NULL_PTR
67
68 Q_ERRORCODE queueIsFull (Queue, BOOLEAN *);
69 // results: If Queue is full, return true;
70 // otherwise, return false
71 // Possible errors:
72 // Q_ERROR_INVALID_QUEUE
73 // Q_ERROR_NULL_PTR
74
75 Q_ERRORCODE queueIsEmpty (Queue, BOOLEAN *);
76 // results: If queue is empty, return true;
77 // otherwise, return false
78 // Possible errors:
79 // Q_ERROR_INVALID_QUEUE
80 // Q_ERROR_NULL_PTR
81
82 /*************************************************************************
83 *                Inserting and retrieving values
84 *************************************************************************/
85 Q_ERRORCODE queueEnqueue (QueuePtr, Q_DATATYPE);
86 // requires: Queue is not full
87 // results: Insert the element at the back of the queue
88 // Possible Errors:
89 // Q_ERROR_INVALID_QUEUE
90 // Q_ERROR_FULL_QUEUE
91
92 Q_ERRORCODE queueDequeue (QueuePtr, Q_DATATYPE *);
93 // requires: Queue is not empty
94 // results: The top element is deleted and its
95 // predecessor becomes the top element.
96 // The deleted element is returned through the argument
97 // list.
98 // Possible Errors:
99 // Q_ERROR_INVALID_QUEUE
100 // Q_ERROR_NULL_PTR
101 // Q_ERROR_EMPTY_LIST
102
103 /*************************************************************************
104 *                        Peek Operations
105 *************************************************************************/
106 Q_ERRORCODE queuePeek (QueuePtr, Q_DATATYPE *);
107 // requires: Queue is not empty
108 // results: The value of the top element is
109 // returned through the argument list
110 // IMPORTANT: Do not remove element from the queue
111 // Possible Errors:
112 //  Q_ERROR_INVALID_QUEUE
113 //  Q_ERROR_NULL_PTR
114 //  Q_ERROR_EMPTY_QUEUE
115
116 /*************************************************************************
117 *                        Error Output Operations
118 *************************************************************************/
119 const char* queueErrorString(Q_ERRORCODE theError);
120 // requires: valid error code
121 // returns a const char * to an entry in the ERRORMSGS array defined in pQueue.c
122 // if theError does not contain a valid error message, "NO ERROR" is returned.
123 // Possible Errors:
124 // None.
125
126 #endif /* QUEUE_H_ */
```