```
1 /*****************************************************************************
2  File name:  pQueue.h
3  Author:     $Author: chadd $
4  Date:       Oct 7, 2011
5  Class:
6  Assignment:
7  Purpose:
8  RevisionID:  $Id: pQueue.h 92 2011-10-08 06:54:39Z chadd $
9  *****************************************************************************/
10
11
12 #ifndef PQUEUE_H_
13 #define PQUEUE_H_
14
15 #include "../../CS300StaticList/include/list.h"
16
17
18 typedef short int PQ_ERRORCODE;
19
20 // BOOLEAN is provided by list.h
21 //typedef short int BOOLEAN;
22 //#define TRUE 1
23 //#define FALSE 0
24
25 // Queue error codes for each function to use
26 #define PQ_NO_ERROR               0
27
28 // Queue create failed
29 #define PQ_ERROR_NO_QUEUE_CREATE  -1
30
31 // user tried to operate on an empty Queue
32 #define PQ_ERROR_EMPTY_QUEUE      -2
33
34 // user tried to add data to a full Queue
35 #define PQ_ERROR_FULL_QUEUE       -3
36
37 // user tried to operate on an invalid Queue. An invalid
38 // Queue may be a NULL QueuePtr or contain an invalid List
39 #define PQ_ERROR_INVALID_QUEUE    -9
40
41 // user provided a NULL pointer to the function (other than the QueuePtr)
42 #define PQ_ERROR_NULL_PTR         -10
43
44
45
46 typedef struct PriorityQueue
47 {
48   List sTheList;
49 } PriorityQueue;
50
51 typedef PriorityQueue * PriorityQueuePtr;
52
53 /*****************************************************************************
54 *                    Allocation and Deallocation
55 *****************************************************************************/
56 PQ_ERRORCODE pqueueCreate (PriorityQueuePtr);
57 // results: If Queue can be created, then Queue exists and
58 // is empty returning Q_NO_ERROR; otherwise,
59 // PQ_ERROR_NO_QUEUE_CREATE is returned
60
61
62 PQ_ERRORCODE pqueueTerminate (PriorityQueuePtr);
63 // results: Queue no longer exists
64 // PQ_ERROR_INVALID_QUEUE
```

```
65
66 /*************************************************************************
67 *                 Checking number of elements in queue
68 *************************************************************************/
69 PQ_ERRORCODE pqueueSize (PriorityQueue, int *);
70 // results: Returns the number of elements in the Queue
71 // Possible errors:
72 // PQ_ERROR_INVALID_QUEUE
73 // PQ_ERROR_NULL_PTR
74
75 PQ_ERRORCODE pqueueIsFull (PriorityQueue, BOOLEAN *);
76 // results: If Queue is full, return true;
77 // otherwise, return false
78 // Possible errors:
79 // PQ_ERROR_INVALID_QUEUE
80 // PQ_ERROR_NULL_PTR
81
82 PQ_ERRORCODE pqueueIsEmpty (PriorityQueue, BOOLEAN *);
83 // results: If queue is empty, return true;
84 // otherwise, return false
85 // Possible errors:
86 // PQ_ERROR_INVALID_QUEUE
87 // PQ_ERROR_NULL_PTR
88
89 /*************************************************************************
90 *                 Inserting and retrieving values
91 *************************************************************************/
92 PQ_ERRORCODE pqueueEnqueue (PriorityQueuePtr, Q_DATATYPE, int);
93 // requires: Queue is not full
94 // results: Insert the element at the back of the queue
95 // Possible Errors:
96 // PQ_ERROR_INVALID_QUEUE
97 // PQ_ERROR_FULL_QUEUE
98
99 PQ_ERRORCODE pqueueDequeue (PriorityQueuePtr, Q_DATATYPE *, int*);
100 // requires: Queue is not empty
101 // results: The top element is deleted and its
102 // predecessor becomes the top element.
103 // The deleted element is returned through the argument
104 // list.
105 // Possible Errors:
106 // PQ_ERROR_INVALID_QUEUE
107 // PQ_ERROR_NULL_PTR
108 // PQ_ERROR_EMPTY_LIST
109
110 /*************************************************************************
111 *                        Peek Operations
112 *************************************************************************/
113 PQ_ERRORCODE pqueuePeek (PriorityQueuePtr, Q_DATATYPE *, int*);
114 // requires: Queue is not empty
115 // results: The value of the top element is
116 // returned through the argument list
117 // IMPORTANT: Do not remove element from the queue
118 // Possible Errors:
119 // PQ_ERROR_INVALID_QUEUE
120 // PQ_ERROR_NULL_PTR
121 // PQ_ERROR_EMPTY_QUEUE
122
123 PQ_ERRORCODE pqueueChangePriority(PriorityQueuePtr psQueue, int change);
124 // requires a non-empty Queue
125 // results: add the value change to each priority in the queue.
126 // the value change may be positive or negative
127 // Possible Errors:
128 // PQ_ERROR_INVALID_QUEUE
```

```
129 // PQ_ERROR_EMPTY_QUEUE
130
131 /*****************************************************************************
132 *                          Error Output Operations
133 *****************************************************************************/
134 const char* pqueueErrorString(PQ_ERRORCODE theError);
135 // requires: valid error code
136 // returns a const char * to an entry in the ERRORMSGS array defined in pQueue.c
137 // if theError does not contain a valid error message, "NO ERROR" is returned.
138 // Possible Errors:
139 // None.
140
141
142 #endif /* PQUEUE_H_ */
143
144
```