```
/*****************************************************************************
 File name:  pQueue.h
 Author:     Shereen Khoja
 Date:       Oct 14, 2012
 Class:
 Assignment:
 Purpose:
 *****************************************************************************/


#ifndef PQUEUE_H_
#define PQUEUE_H_

#include "../../CS300DynamicList/include/list.h"

typedef short int PQ_ERRORCODE;

// BOOLEAN is provided by list.h
//typedef short int BOOLEAN;
//#define TRUE 1
//#define FALSE 0

// Queue error codes for each function to use
#define PQ_NO_ERROR               0

// Queue create failed
#define PQ_ERROR_NO_QUEUE_CREATE  -1

// user tried to operate on an empty Queue
#define PQ_ERROR_EMPTY_QUEUE      -2

// user tried to add data to a full Queue
#define PQ_ERROR_FULL_QUEUE       -3

// user tried to operate on an invalid Queue. An invalid
// Queue may be a NULL QueuePtr or contain an invalid List
#define PQ_ERROR_INVALID_QUEUE    -9

// user provided a NULL pointer to the function (other than the QueuePtr)
#define PQ_ERROR_NULL_PTR         -10



typedef struct PriorityQueue
{
  List sTheList;
} PriorityQueue;
```

```
typedef PriorityQueue * PriorityQueuePtr;

/*************************************************************************
*                    Allocation and Deallocation
*************************************************************************/
PQ_ERRORCODE pqueueCreate (PriorityQueuePtr);
// results: If Queue can be created, then Queue exists and
// is empty returning Q_NO_ERROR; otherwise,
// PQ_ERROR_NO_QUEUE_CREATE is returned


PQ_ERRORCODE pqueueTerminate (PriorityQueuePtr);
// results: Queue no longer exists
// PQ_ERROR_INVALID_QUEUE

/*************************************************************************
*                  Checking number of elements in queue
*************************************************************************/
PQ_ERRORCODE pqueueSize (PriorityQueue, int *);
// results: Returns the number of elements in the Queue
// Possible errors:
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_NULL_PTR

PQ_ERRORCODE pqueueIsEmpty (PriorityQueue, BOOLEAN *);
// results: If queue is empty, return true;
// otherwise, return false
// Possible errors:
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_NULL_PTR

/*************************************************************************
*                  Inserting and retrieving values
*************************************************************************/
PQ_ERRORCODE pqueueEnqueue (PriorityQueuePtr, Q_DATATYPE);
// requires: Queue is not full
// results: Insert the element at the back of the queue
// Possible Errors:
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_FULL_QUEUE

PQ_ERRORCODE pqueueDequeue (PriorityQueuePtr, Q_DATATYPE *);
// requires: Queue is not empty
// results: The top element is deleted and its
// predecessor becomes the top element.
// The deleted element is returned through the argument
// list.
// Possible Errors:
```

```
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_NULL_PTR
// PQ_ERROR_EMPTY_LIST

/*****************************************************************************
*                          Peek Operations
*****************************************************************************/
PQ_ERRORCODE pqueuePeek (PriorityQueuePtr, Q_DATATYPE *);
// requires: Queue is not empty
// results: The value of the top element is
// returned through the argument list
// IMPORTANT: Do not remove element from the queue
// Possible Errors:
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_NULL_PTR
// PQ_ERROR_EMPTY_QUEUE

PQ_ERRORCODE pqueueChangePriority(PriorityQueuePtr psQueue, int change);
// requires: Queue is not empty
// results: The priority of all elements is increased
// by change
// Possible Errors:
// PQ_ERROR_INVALID_QUEUE
// PQ_ERROR_NULL_PTR
// PQ_ERROR_EMPTY_QUEUE


/*****************************************************************************
*                          Error Output Operations
*****************************************************************************/
const char* pqueueErrorString(PQ_ERRORCODE theError);


#endif /* PQUEUE_H_ */
```