```
/*
 * CS300 EXAM REVIEW TOPICS
 *
 *
 * Below you will find datatypes and functions for a List,
 * Priority Queue, Queue, and Binary Tree.
 *
 * 1) You may be asked to explain certain things about said data
 *            structures, build different data structures using one or
 *            more of the existing data structures below, or modify existing
 *            data structures.
 *
 *    a) How would you build a dynamic stack out of a list?
 *    b) How would you build a dynamic array out of a list?
 *    c) Could you build a dynamic array out of a priority queue with
 *             reasonable functionality?
 *    d) What are the computing complexities of 1) b) versus a regular
 *             array for various operations such as updating an arbitrary
 *             element, inserting an element into the array, deleting an
 *             element from the array, sorting an array, searching an ordered
 *             array
 *
 * 2) Advantages & disadvantages of arrays, linked lists, stacks,
 *            queues, trees
 *
 * 3) Tree terminology ... root, parent, child, leaf, siblings,
 *            ancestors, descendents, subtree, level, depth, height
 *
 * 4) Binary Tree vs Binary Search Tree vs AVL Tree including computing
 *      complexities for various operations
 *
 * 5) Traversals ... preorder, postorder, inorder
 *
 * 6) Tree Problems
 *
 *            a) Draw the unique binary tree that has the inorder traversal
 *                   X, Z, C, T, G, P, R and the preorder traversal
 *                   C, Z, X, R, G, T, P.
 *            b) Write a function treeCopy that makes an identical copy of
 *                   a binary tree
 *            c) Write a function that returns the number of internal nodes
 *                   of a binary tree
 *
 * 7) Generic Programming ... void *, memcpy, memcmp,
 *            address arithmetic, function pointers, passing functions as
 *            arguments
 *
 * 8) Generic Problems
 *
 *            a) Our list implementation is pretty good but what is the main
 *                   drawback to our representation? What change(s) would you
 *                   make?
 *            b) Consider the function prototype
 *                   void qsort(void *base, size_t nmemb, size_t size,
 *                   int (*compar)(const void *, const void *));
 *                   Make sure you can explain the use of each parameter
```

```
 * /

/*****************************************************************************
 * List Datatypes and Functions
 * ***************************************************************************/

typedef short int ERRORCODE;

typedef struct Q_DATATYPE
{
       int intValue;
       int priority;
}Q_DATATYPE;

typedef struct DATATYPE
{
       Q_DATATYPE data;

       union
  {
       char charValue;
       unsigned int intValue;
    float floatValue;
  };
  unsigned short whichOne;
} DATATYPE;

typedef struct ListElement* ListElementPtr;
typedef struct ListElement
{
       DATATYPE data;
       ListElementPtr psNext;
       ListElementPtr psPrev;
} ListElement;

typedef struct List* ListPtr;
typedef struct List
{
       ListElementPtr psHead;
       ListElementPtr psLast;
       ListElementPtr psCurrent;
       int numElements;
} List;

// List functions
ERRORCODE lstCreate (ListPtr );
ERRORCODE lstDispose (ListPtr );

ERRORCODE lstSize (ListPtr, int *);
ERRORCODE lstIsFull (ListPtr, bool *);
ERRORCODE lstIsEmpty (ListPtr, bool *);

ERRORCODE lstPeek (ListPtr, DATATYPE *);
ERRORCODE lstPeekPrev (ListPtr, DATATYPE *);
ERRORCODE lstPeekNext (ListPtr, DATATYPE *);

ERRORCODE lstFirst(ListPtr, DATATYPE *);
```

```
ERRORCODE lstLast(ListPtr, DATATYPE *);
ERRORCODE lstNext(ListPtr, DATATYPE *);
ERRORCODE lstPrev(ListPtr, DATATYPE *);

ERRORCODE lstDeleteCurrent (ListPtr, DATATYPE *);
ERRORCODE lstInsertAfter (ListPtr, DATATYPE);
ERRORCODE lstInsertBefore (ListPtr, DATATYPE);
ERRORCODE lstUpdateCurrent (ListPtr, DATATYPE);

ERRORCODE lstHasNext (ListPtr, bool *);
ERRORCODE lstHasPrev(ListPtr, bool *);
ERRORCODE lstHasCurrent (ListPtr, bool *);

/*****************************************************************************
 * Priority Queue Datatypes and Functions
 * ***************************************************************************/
typedef short int PQ_ERRORCODE;

typedef struct PriorityQueue* PriorityQueuePtr;
typedef struct PriorityQueue
{
	List sTheList;
} PriorityQueue;

PQ_ERRORCODE pqueueCreate (PriorityQueuePtr);
PQ_ERRORCODE pqueueTerminate (PriorityQueuePtr);

PQ_ERRORCODE pqueueSize (PriorityQueue, int *);
PQ_ERRORCODE pqueueIsEmpty (PriorityQueue, bool *);
PQ_ERRORCODE pqueueIsFull (PriorityQueue, bool *);
PQ_ERRORCODE pqueueEnqueue (PriorityQueuePtr, Q_DATATYPE);
PQ_ERRORCODE pqueueDequeue (PriorityQueuePtr, Q_DATATYPE *);
PQ_ERRORCODE pqueuePeek (PriorityQueuePtr, Q_DATATYPE *);
PQ_ERRORCODE pqueueChangePriority(PriorityQueuePtr psQueue, int change);

/*****************************************************************************
 * Queue Datatypes and Functions
 * ***************************************************************************/

typedef short int Q_ERRORCODE;

typedef struct Queue* QueuePtr;
typedef struct Queue
{
	PriorityQueue sTheQueue;
} Queue;

Q_ERRORCODE queueCreate (QueuePtr);
Q_ERRORCODE queueTerminate (QueuePtr);
Q_ERRORCODE queueSize (Queue, int *);
Q_ERRORCODE queueIsEmpty (Queue, bool *);
Q_ERRORCODE queueIsFull (Queue, bool *);
Q_ERRORCODE queueEnqueue (QueuePtr, Q_DATATYPE);
Q_ERRORCODE queueDequeue (QueuePtr, Q_DATATYPE *);
Q_ERRORCODE queuePeek (QueuePtr, Q_DATATYPE *);

/*****************************************************************************
 * Tree Datatypes and Functions
```

```
 * ************************************************************************/
typedef struct TreeNode *TreeNodePtr;
typedef struct TreeNode
{
       TreeNodePtr psLeftChild, psRightChild;
       int data;
} TreeNode;

void bstCreate (TreeNodePtr *hsRoot);
void bstPrint (TreeNodePtr psRoot, unsigned short numSpaces,

                            unsigned short level);
void bstInsertIterative (TreeNodePtr *hsRoot, int data);
int bstSize (TreeNodePtr psRoot);
void bstTerminate (TreeNodePtr *hsRoot);
bool bstIsIn (TreeNodePtr psRoot, int data);
TreeNodePtr bstFindNode (TreeNodePtr psRoot, int data);
TreeNodePtr bstFindParent (TreeNodePtr psRoot, TreeNodePtr psNode);
int bstFindLevel (TreeNodePtr psRoot);
void bstInsertRecusive (TreeNodePtr *hsRoot, int data);
```