Assignment--Right Threaded-BST

```
struct rightThreadedBSTNode
{
     int key;
     int size;    // Total no of nodes in the subtree rooted at the node.
     int rightThread; // 1, if right link is a thread to its inorder successor
     struct rightThreadedBSTNode *leftChild;
     struct rightThreadedBSTNode *rightChild;
};
typedef struct  rightThreadedBSTNode  rTBSTNode;
typedef struct  rightThreadedBSTNode* rTBSTNodePtr;
```

void createRightThreadedBST(rTBSTNodePtr *root){ *rTBSTNodePtr = NULL;}
rTBSTNodePtr getRightThreadedTreeNode() //Allocate a node dynamically

void displayRightThreadedBST(rTBSTNodePtr root, char *fileName)
// Use graphviz to display tree graphically

rTBSTNodePtr rtbstSearch(rTBSTNodePtr root, int data) - O(h)
// returns the pointer of the node having key value equal to data (successful search) or returns NULL

[15 points] void inorder(rTBSTNodePtr root, int *List);
// Non-recursive and without using stack

[10 points] int rtbstInsert(rTBSTNodePtr *root, int data) - O(h)

[40 points] int rtbstDelete(rTBSTNodePtr *root, int data) - O(h)

[20 points] rTBSTNodePtr kthElement(rTBSTNodePtr *root, int k) - O(h)
// returns the pointer of the node having the k-th smallest key value ow returns NULL

Note: Don't change the function definition. No marks for partial implementation.

Assignment--Right Threaded-BST

```
struct rightThreadedBSTNode
{
     int key;
     int size;    // Total no of nodes in the subtree rooted at the node.
     int rightThread; // 1, if right link is a thread to its inorder successor
     struct rightThreadedBSTNode *leftChild;
     struct rightThreadedBSTNode *rightChild;
};
typedef struct  rightThreadedBSTNode  rTBSTNode;
typedef struct  rightThreadedBSTNode* rTBSTNodePtr;
```

void createRightThreadedBST(rTBSTNodePtr *root){ *rTBSTNodePtr = NULL;}
rTBSTNodePtr getRightThreadedTreeNode() //Allocate a node dynamically

void displayRightThreadedBST(rTBSTNodePtr root, char *fileName)
// Use graphviz to display tree graphically

rTBSTNodePtr rtbstSearch(rTBSTNodePtr root, int data) - O(h)
// returns the pointer of the node having key value equal to data (successful search) or returns NULL

[15 points] void inorder(rTBSTNodePtr root, int *List);
// Non-recursive and without using stack

[10 points] int rtbstInsert(rTBSTNodePtr *root, int data) - O(h)

[40 points] int rtbstDelete(rTBSTNodePtr *root, int data) - O(h)

[20 points] rTBSTNodePtr kthElement(rTBSTNodePtr *root, int k) - O(h)
// returns the pointer of the node having the k-th smallest key value ow returns NULL

Note: Don't change the function definition. No marks for partial implementation.

Assignment--Right Threaded-BST

```
struct rightThreadedBSTNode
{
     int key;
     int size;    // Total no of nodes in the subtree rooted at the node.
     int rightThread; // 1, if right link is a thread to its inorder successor
     struct rightThreadedBSTNode *leftChild;
     struct rightThreadedBSTNode *rightChild;
};
typedef struct  rightThreadedBSTNode  rTBSTNode;
typedef struct  rightThreadedBSTNode* rTBSTNodePtr;
```

void createRightThreadedBST(rTBSTNodePtr *root){ *rTBSTNodePtr = NULL;}
rTBSTNodePtr getRightThreadedTreeNode() //Allocate a node dynamically

void displayRightThreadedBST(rTBSTNodePtr root, char *fileName)
// Use graphviz to display tree graphically

rTBSTNodePtr rtbstSearch(rTBSTNodePtr root, int data) - O(h)
// returns the pointer of the node having key value equal to data (successful search) or returns NULL

[15 points] void inorder(rTBSTNodePtr root, int *List);
// Non-recursive and without using stack

[10 points] int rtbstInsert(rTBSTNodePtr *root, int data) - O(h)

[40 points] int rtbstDelete(rTBSTNodePtr *root, int data) - O(h)

[20 points] rTBSTNodePtr kthElement(rTBSTNodePtr *root, int k) - O(h)
// returns the pointer of the node having the k-th smallest key value ow returns NULL

Note: Don't change the function definition. No marks for partial implementation.

Assignment--Right Threaded-BST

```
struct rightThreadedBSTNode
{
     int key;
     int size;    // Total no of nodes in the subtree rooted at the node.
     int rightThread; // 1, if right link is a thread to its inorder successor
     struct rightThreadedBSTNode *leftChild;
     struct rightThreadedBSTNode *rightChild;
};
typedef struct  rightThreadedBSTNode  rTBSTNode;
typedef struct  rightThreadedBSTNode* rTBSTNodePtr;
```

void createRightThreadedBST(rTBSTNodePtr *root){ *rTBSTNodePtr = NULL;}
rTBSTNodePtr getRightThreadedTreeNode() //Allocate a node dynamically

void displayRightThreadedBST(rTBSTNodePtr root, char *fileName)
// Use graphviz to display tree graphically

rTBSTNodePtr rtbstSearch(rTBSTNodePtr root, int data) - O(h)
// returns the pointer of the node having key value equal to data (successful search) or returns NULL

[15 points] void inorder(rTBSTNodePtr root, int *List);
// Non-recursive and without using stack

[10 points] int rtbstInsert(rTBSTNodePtr *root, int data) - O(h)

[40 points] int rtbstDelete(rTBSTNodePtr *root, int data) - O(h)

[20 points] rTBSTNodePtr kthElement(rTBSTNodePtr *root, int k) - O(h)
// returns the pointer of the node having the k-th smallest key value ow returns NULL

Note: Don't change the function definition. No marks for partial implementation.