1. （允許二元樹包含重複數值）修改圖12.19的程式來讓二元樹能夠包含重複的數值。以左子樹節點值的值皆小於等於根節點，並以中序(inorder)輸出。[輸入值: 12, 5, 8, 6, 12, 19, 2, 4](30%)

**2.** （二元樹搜尋）撰寫一個成員函式binaryTreeSearch，以在二元搜尋樹物件中搜尋指定的值。此函式應接受兩個引數，一個是指向二元樹根節點的指標,，另一個是搜尋鍵值。若找到含有搜尋鍵值的節點，函式應傳回指向該節點的指標；否則函式就傳回NULL指標。[輸入值: 12, 15, 8, 6, 17 ,19, 2, 4](30%)

**3.** （逐層走訪二元樹）圖12.19介紹三種走訪二元樹的遞迴方法：中序、前序、後序走訪。這個習題將介紹二元樹的階層順序走訪（level order traversal），在此方法中，節點的數值會由根節點階層開始，一層一層地顯示出來。每個階層的節點是從左往右列印。階層順序走訪並不是遞迴演算法。它使用佇列資料結構來控制節點的輸出。[輸入值: 12, 15, 8, 6, 17 ,19, 2, 4]演算法如下：

1)將根節點加入佇列。

2)當佇列中還有節點時，

取得佇列的下一個節點  
列印節點的值

　若此節點的左指標不為NULL  
將左子節點加入佇列

　若此節點的右指標不為NULL  
將右子節點加入佇列

撰寫函式levelOrder來執行二元樹的階層順序走訪。函式應該以指向二元樹根節點的指標為引數。修改圖12.19的程式來使用這個函式。比較此函式與其它走訪演算法的輸出結果，確認此函式的執行結果是否正確。[請注意：本程式中，你也需修改並使用圖12.13的佇列處理函式。](40%)

1. ***(Duplicate Elimination)*** We’ve seen that duplicate elimination is straightforward when creating a binary search tree. Describe how you would perform duplicate elimination using only a one-dimensional array. Compare the performance of array-based duplicate elimination with the performance of binary-search-tree-based duplicate elimination. We can redefine that the values of nodes in the left subtree are greater than or equal to the value of the root. Please output the result by inorder. [input values: 12, 5, 8, 6, 12, 19, 2, 4] (30%)

2. ***(Binary Tree Search)*** Write function binaryTreeSearch that attempts to locate a specified value in a binary search tree. The function should take as arguments a pointer to the root node of the binary tree and a search key to be located. If the node containing the search key is found, the function should return a pointer to that node; otherwise, the function should return a NULL pointer. [input values: 12, 15, 8, 6, 17 ,19, 2, 4] (30%)

3. ***(Level Order Binary Tree Traversal)*** The program of Fig. 12.19 illustrated three recursive methods of traversing a binary tree—inorder traversal, preorder traversal, and postorder traversal. This exercise presents the **level order traversal** of a binary tree in which the node values are printed level-by-level starting at the root node level. The nodes on each level are printed from left to right. The level order traversal is not a recursive algorithm. It uses the queue data structure to control the output of the nodes. [input values: 12, 15, 8, 6, 17 ,19, 2, 4] The algorithm is as follows:

1) Insert the root node in the queue

2) While there are nodes left in the queue,

Get the next node in the queue

Print the node’s value

If the pointer to the left child of the node is not NULL

Insert the left child node in the queue

If the pointer to the right child of the node is not NULL

Insert the right child node in the queue.

Write function levelOrder to perform a level order traversal of a binary tree. The function should take as an argument a pointer to the root node of the binary tree. Modify the program of Fig. 12.19 to use this function. Compare the output from this function to the outputs of the other traversal algorithms to see that it worked correctly. [*Note:* You’ll also need to modify and incorporate the queue-processing functions of Fig. 12.13 in this program.] (40%)