Section F – Binary Search Trees [Answer 1 Specified Qn from this Section]

Information: Program templates for questions are available in APAS system. You must use them to implement your functions.

1. (**levelOrderTraversal**) Write an iterative C function `levelOrderTraversal` prints a level-by-level traversal of the binary tree using a **queue**, starting at the root node level. Note that you should **<u>only</u>** use `enqueue()` or `dequeue()` operations when you add or remove integers from the queue. Remember to empty the queue at the beginning, if the queue is not empty.

   The function prototype is given as follows:

   ```
   void levelOrderIterative(BSTNode *root);
   ```

   For example, for the binary tree in *Figure 1*, the level order tree traversal is: **20, 15, 50, 10, 18, 25, 80**.

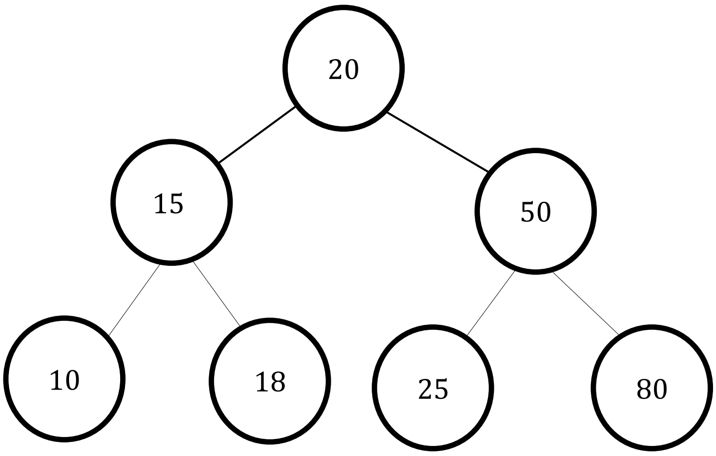

Figure 1

2. **(inOrderIterative)** Write an iterative C function `inOrderIterative()` that prints the in-order traversal of a binary search tree using **a stack**. Note that you should **<u>only</u>** use `push()` or `pop()` operations when you add or remove integers from the stack. Remember to empty the stack at the beginning, if the stack is not empty.

   The function prototype is given as follows:

   ```
   void inOrderIterative(BSTNode *root);
   ```

   For example, for the binary tree in *Figure 2*, the iterative inorder traversal is: **10, 15, 18, 20, 50.**

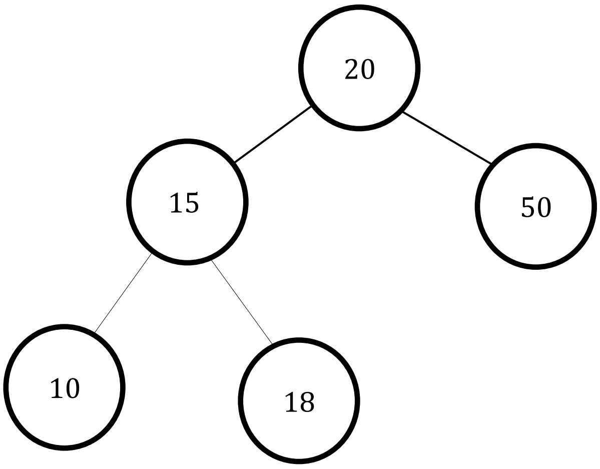

Figure 2

3. **(preOrderIterative)** Write an iterative C function `preOrderIterative()` that prints the pre-order traversal of a binary search tree using **a stack**. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stack. Remember to empty the stack at the beginning, if the stack is not empty.

The function prototype is given as follows:

```
void preOrderIterative(BSTNode *root);
```

For example, for the binary tree in *Figure 3*, the iterative preorder tree traversal is: **20, 15, 10, 18, 50, 25, 80.**

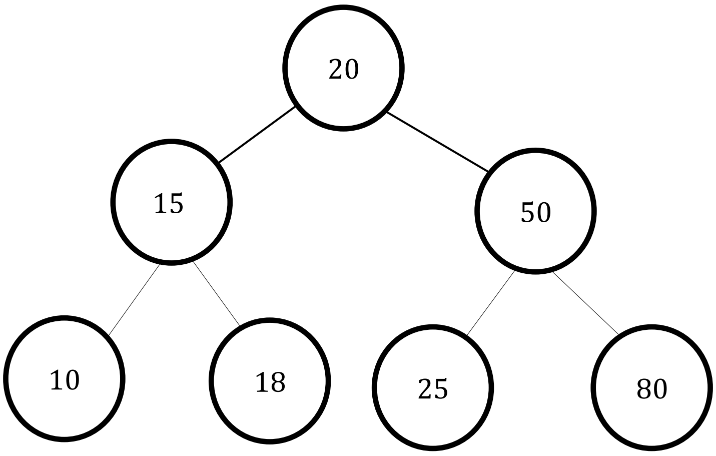

Figure 3

4. **(postOrderIterativeS1)** Write an iterative C function `postOrderIterativeS1()` that prints the post-order traversal of a binary search tree using **a stack**. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stack. Remember to empty the stack at the beginning, if the stack is not empty.

The function prototype is given as follows:

```
void postOrderIterativeS1(BSTNode *node);
```

For example, for the binary tree in *Figure 4*, the iterative postorder tree traversal is: **10, 18, 15, 25, 80, 50, 20.**

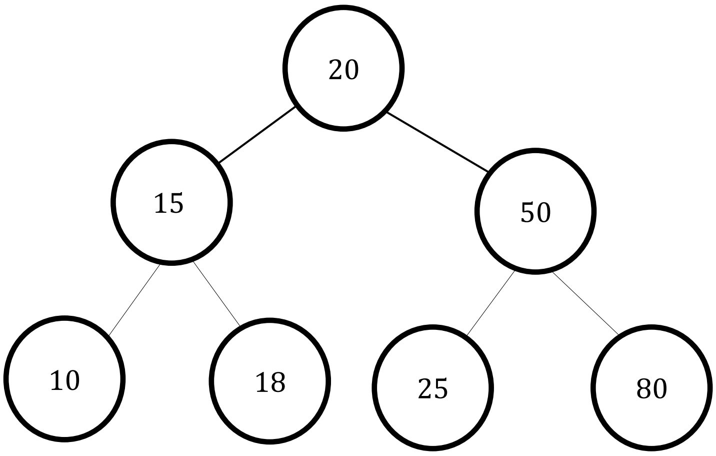

Figure 4

5. **(postOrderIterativeS2)** Write an iterative C function `postOrderIterativeS2()` that prints the post-order traversal of a binary search tree using **two stacks**. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stacks. Remember to empty the stacks at the beginning, if the stacks are not empty.

The function prototype is given as follows:

```
      void postOrderIterativeS2(BSTNode *root);
```

For example, for the binary tree in *Figure 5*, the iterative postorder tree traversal is: **10, 18, 15, 25, 80, 50, 20.**

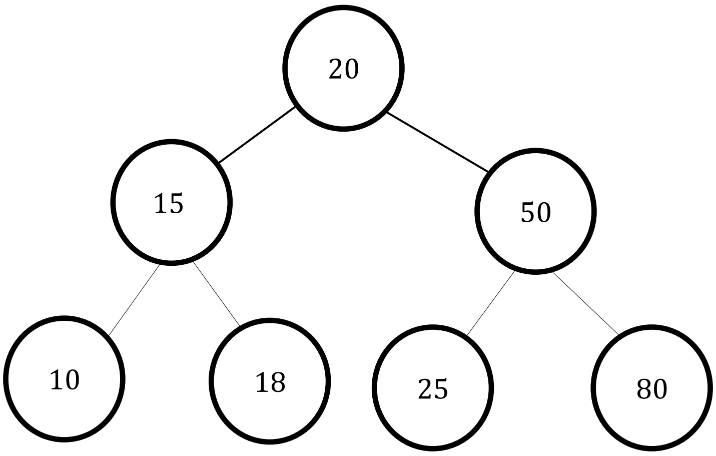

Figure 5