Data Structures and Algorithms
Assignment 2: Binary Trees and Binary Search Trees

Program templates for questions 1-4 are given as separated files. You must use them to implement your functions. You need to submit your code to the **https://www.hackerearth.com** (Email invitation will be sent to your school account).. Deadline for program submission:**[07/03/2025]**.

1. (**identical**) Write a recursive Python function `identical()` to determine whether two binary trees are structurally identical, assuming the two binary trees as `tree1` and `tree2`. This function returns TRUE if two binary trees are structurally identical; otherwise, it returns FALSE. Note that two binary trees are structurally identical if they are both empty or if they are both non-empty and the left and the right subtrees are similar (**they are made of nodes with the same values and arranged in the same way**).

   The function prototype is given as follows:

   ```
   def identical (tree1: BTNode, tree2: BTNode)
   ```

   For example, if the given two trees are tree 1 (**1, 3, 2, 5, 4, 7, 8)** and tree 2 (**1, 3, 2, 5, 4, 7, 8),** as shown in Figure 1, then, tree 1 and tree 2 are **structurally identical**.

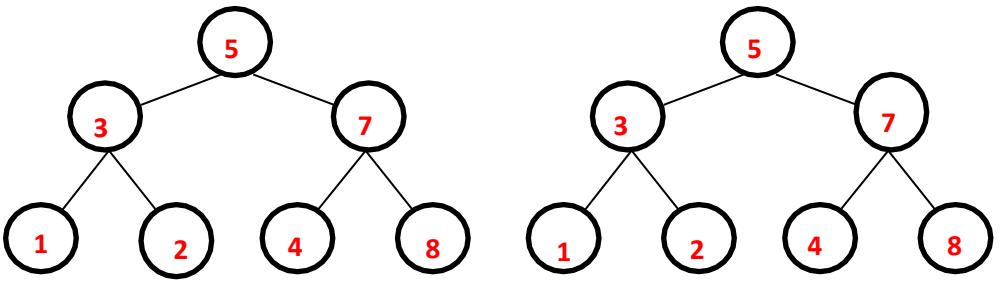

Figure 1: tree1 (left) and tree 2 (right)

A sample input and output session is given below:

```
1: Create a binary tree1.
2: Create a binary tree2.
3: Check whether two trees are structurally identical.
0: Quit;
Please input your choice(1/2/3/0): 1
Creating tree1:
Input an integer that you want to add to the binary tree. Any Alpha value
will be treated as NULL.

Enter an integer value for the root: 5
Enter an integer value for the Left child of 5: 3
Enter an integer value for the Right child of 5: 7
Enter an integer value for the Left child of 3: 1
Enter an integer value for the Right child of 3: 2
Enter an integer value for the Left child of 1: a
Enter an integer value for the Right child of 1: a
Enter an integer value for the Left child of 2: a
```

```
Enter an integer value for the Right child of 2: a
Enter an integer value for the Left child of 7: 4
Enter an integer value for the Right child of 7: 8
Enter an integer value for the Left child of 4: a
Enter an integer value for the Right child of 4: a

Enter an integer value for the Left child of 8: a
Enter an integer value for the Right child of 8: a
The resulting tree1 is: 1 3 2 5 4 7 8
Please input your choice(1/2/3/0): 2

Creating tree2:
Input an integer that you want to add to the binary tree. Any Alpha value
will be treated as NULL.

Enter an integer value for the root: 5
Enter an integer value for the Left child of 5: 3
Enter an integer value for the Right child of 5: 7
Enter an integer value for the Left child of 3: 1
Enter an integer value for the Right child of 3: 2
Enter an integer value for the Left child of 1: a
Enter an integer value for the Right child of 1: a
Enter an integer value for the Left child of 2: a
Enter an integer value for the Right child of 2: a
Enter an integer value for the Left child of 7: 4
Enter an integer value for the Right child of 7: 8
Enter an integer value for the Left child of 4: a
Enter an integer value for the Right child of 4: a
Enter an integer value for the Left child of 8: a
Enter an integer value for the Right child of 8: a
The resulting tree2 is: 1 3 2 5 4 7 8

Please input your choice(1/2/3/0): 3
Both trees are structurally identical.

Please input your choice(1/2/3/0): 0
```

2. **(inorderIterative)** Write an iterative Python function `inorderIterative()` that prints the in-order traversal of a binary search tree using **only one temporary stack** inside the `inorderIterative()` function. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stack.

The function prototype is given as follows:

```
def inorderIterative(root: BSTNode):
```

Let us consider the below tree for example.

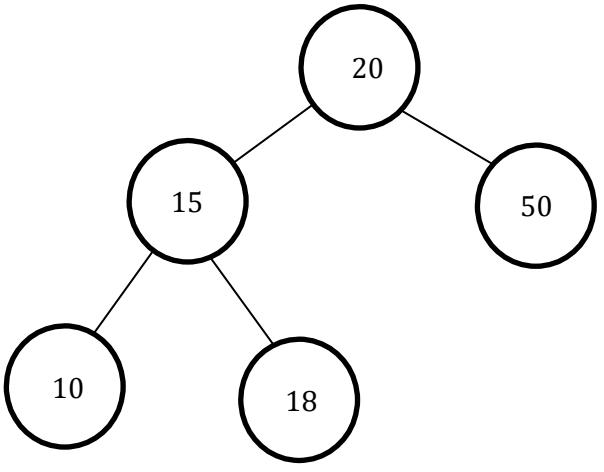

Figure 2: Iterative Inoder Tree Traversal: **10 15 18 20 50**

Following is the detailed algorithm:

```
1) Create an empty stack S.
2) Initialize current node as root
3) Push the current node to S and set current =
   current->left until current is NULL
4) If current is NULL and stack is not empty then
     a) Pop the top item from stack
     b) Print the popped item, set current = popped_item->right
     c) Go to step 3.
5) If current is NULL and stack is empty then you are done
```

Some sample inputs and outputs are given as follows:

```
1: Insert an integer into the binary search tree;
2: Print the in-order traversal of the binary search tree;
0: Quit;

Please input your choice(1/2/0): 1
Input an integer that you want to insert into the Binary Search Tree: 20
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the Binary Search Tree: 15
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the Binary Search Tree: 50
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the Binary Search Tree: 10
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the Binary Search Tree: 18

Please input your choice(1/2/0): 2
The resulting in-order traversal of the binary search tree is: 10 15 18 20 50

Please input your choice(1/2/0): 0
```

3. **(postOrderIterativeS1)** Write an iterative Python function `postOrderIterativeS1()` that prints the post-order traversal of a binary search tree using **only one temporary stack** inside the `postOrderIterativeS1()` function. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stack.

   The function prototype is given as follows:

   ```
   def postOrderIterativeS1(node: BSTNode):
   ```

Let us consider the below tree for example

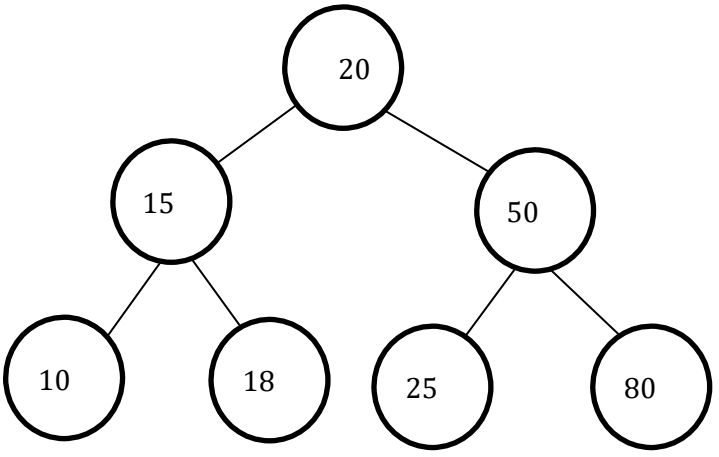

Figure 3: Iterative Postorder Traversal: **10 18 15 25 80 50 20**

Some sample inputs and outputs are given as follows:

```
1: Insert an integer into the binary search tree
2: Print the post-order traversal of the binary search tree
0: Quit

Please input your choice(1/2/0): 1
Input an integer to insert: 20

Please input your choice(1/2/0): 1
Input an integer to insert: 15

Please input your choice(1/2/0): 1
Input an integer to insert: 50

Please input your choice(1/2/0): 1
Input an integer to insert: 10

Please input your choice(1/2/0): 1
Input an integer to insert: 18

Please input your choice(1/2/0): 1
Input an integer to insert: 25

Please input your choice(1/2/0): 1
Input an integer to insert: 80

Please input your choice(1/2/0): 2
Post-order traversal: 10 18 15 25 80 50 20

Please input your choice(1/2/0): 0
```

4. **(postOrderIterativeS2)** Write an iterative Python function `postOrderIterativeS2()` that prints the post-order traversal of a binary search tree using **no more and no less than two temporary stacks** inside the `postOrderIterativeS2()` function. Note that you should **only** use `push()` or `pop()` operations when you add or remove integers from the stacks.

   The function prototype is given as follows:

   ```
   def postOrderIterativeS2(root: BSTNode)
   ```

   Let us consider the below tree for example

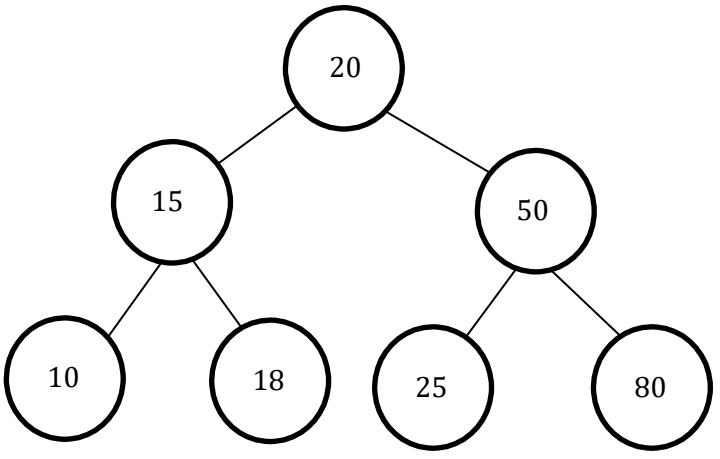

Figure 4: Iterative Postorder Tree Traversal: **10 18 15 25 80 50 20**

Some sample inputs and outputs are given as follows:

```
1: Insert an integer into the binary search tree
2: Print the post-order traversal of the binary search tree
0: Quit

Please input your choice(1/2/0): 1
Input an integer to insert: 20

Please input your choice(1/2/0): 1
Input an integer to insert: 15

Please input your choice(1/2/0): 1
Input an integer to insert: 50

Please input your choice(1/2/0): 1
Input an integer to insert: 10

Please input your choice(1/2/0): 1
Input an integer to insert: 18

Please input your choice(1/2/0): 1
Input an integer to insert: 25

Please input your choice(1/2/0): 1
Input an integer to insert: 80

Please input your choice(1/2/0): 2
Post-order traversal: 10 18 15 25 80 50 20

Please input your choice(1/2/0): 0
```