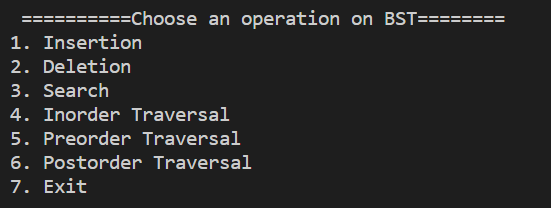
**WEEK 8&9**

**AIM: Insertion, deletion, and traversal in the binary search tree (BST)**

In this exercise, I made a menu-driven program where I gave the user 7 choices. First, 3 choices are basic operations on BST i.e insertion, deletion, search, other 3 choices are different traversals of BST i.e preorder, inorder, postorder and the last choice is to exit the program.



**VARIABLES USED**

The root of the BST: root

Name of the node: node

Data in node: data

Pointer to a right child: right

Pointer to a left child: left

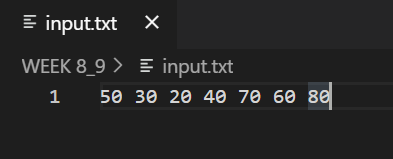
Value of the node to be deleted: key

Pointer for traversing BST: curr

Helper variable: choice,n,fptr

I

The initial content of input.txt file for given BST:

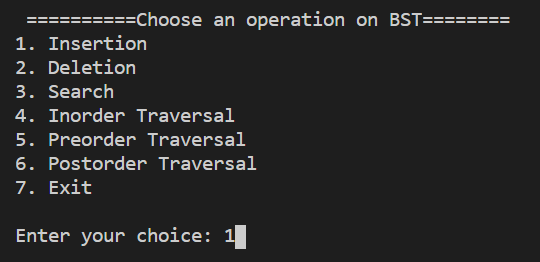


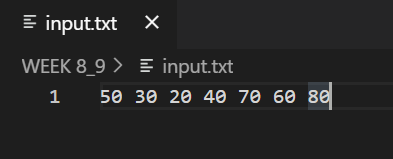
1. **Insertion:** This function is used to insert the node to the BST.

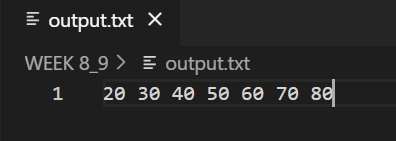
**Algorithm:**

1. Allocate the memory for the node.
2. Set the data part to the value and set the left and right pointer of the node, point to NULL.
3. If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
4. Else, check if the item is less than the root element of the tree, if this is true, then recursively perform insertion operation with the left of the root.
5. If this is false, then perform insertion recursively with the right sub-tree of the root.

**Sample Input**: 50 30 20 40 70 60 80







**Time complexity:** Average: O(h) // h: height of the bst

Worst: O(n) // n: total no. of nodes in the BST

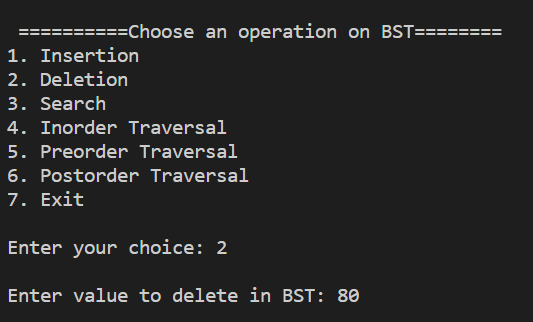
1. **Deletion**: This function is used to delete a node by its key value from the BST.

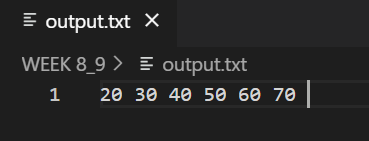
**Algorithm:**

1. If the root is NULL then return.
2. If data of the current node is less than the key to be deleted then perform deletion recursively on the right subtree of the node.
3. If data of the current node is greater than the key to be deleted then perform deletion recursively on the left subtree of the node.
4. If the key to be deleted is equal to the current node data:
5. If the node is a leaf (both left and right will be NULL), remove the node directly and free its memory.
6. If the node has an only right child (left will be NULL), make the node point to the right node and free the node.
7. If the node has an only left child (right will be NULL), make the node point to the left node and free the node.
8. If the node has both left and right child,
   1. 1.find the inorder successor say, temp
   2. 2.make node->data = temp->data
   3. 3.Again delete the temp node.

**Time complexity:** Average: O(h) // h: height of the bst

Worst: O(n) // n: total no. of nodes in the BST





**Sample Input:** 80

**Sample Output:** (\* 80 deleted from the BST \*)

1. **Search:** This function is used to search a key in the given BST.

Algorithm:

1. If the root is NULL then return -1 to indicate key not found.
2. If data of the current node is less than the key to be deleted then perform search recursively on the right subtree of the node.
3. If data of the current node is greater than the key to be deleted then perform search recursively on the left subtree of the node.
4. If the key to be deleted is equal to the current node data then return 1 to indicate found.

**Time complexity:** Average: O(h) // h: height of the bst

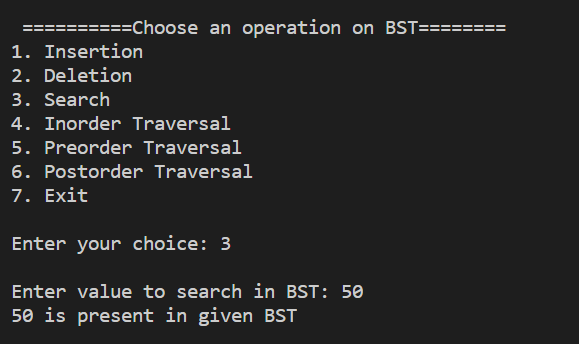
Worst: O(n) // n: total no. of nodes in the BST

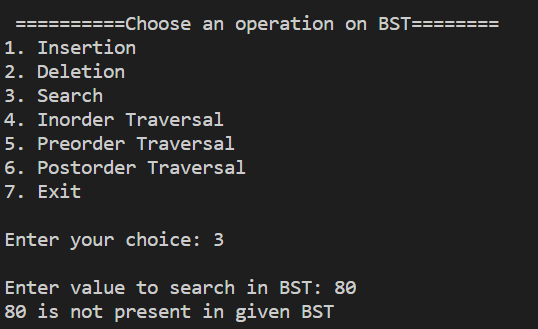
**Sample input:** 50

**Sample Output:** 50 is present in the given BST

**Sample input:** 80

**Sample Output:** 80 is not present in the given BST



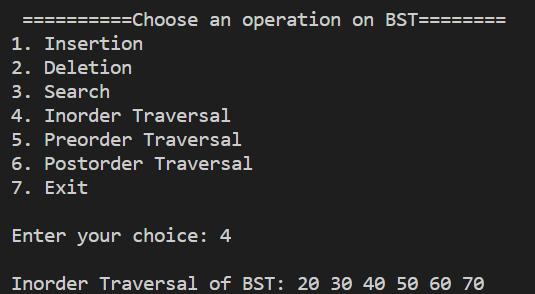


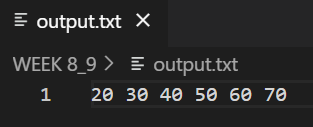
**4. Inorder Traversal**

**Algorithm:**

1. If the root is null then return
2. Recursively call the function passing the left child of the current node.
3. Print the data of the current node.
4. Recursively call the function passing the right child of the current node.

**Time Complexity:** O(n)



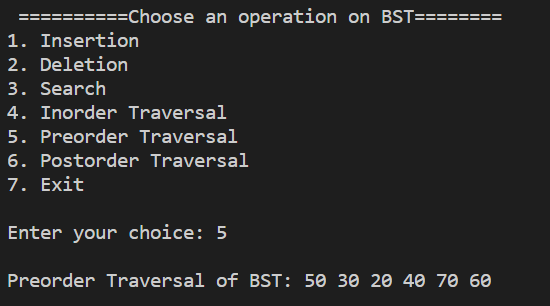


**5. Preorder Traversal**

**Algorithm:**

1. If the root is null then return
2. Print the data of the current node.
3. Recursively call the function passing the left child of the current node.
4. Recursively call the function passing the right child of the current node.

**Time Complexity:** O(n)



**6. Postorder Traversal**

**Algorithm:**

1. If the root is null then return
2. Recursively call the function passing the left child of the current node.
3. Recursively call the function passing the right child of the current node.
4. Print the data of the current node.

**Time Complexity:** O(n)

