Batch: E-2 **Roll No.: 16010123325**

Experiment / assignment / tutorial No. 7

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of BST & Binary tree traversal techniques.

Objective: To Understand and Implement Binary Search Tree along with Insertion, Deletion and Preorder, Postorder and Inorder Traversal Techniques.

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:
A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

A binary tree is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

A Binary Search Tree is a node-based binary tree data structure in which the left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. The left and right subtree each must also be a binary search tree.

Related Theory: -
Algorithm: Preorder Traversal of BST

1. Start at the root node
2. Visit the root node and print its value
3. Recursively traverse the left subtree
4. Recursively traverse the right subtree

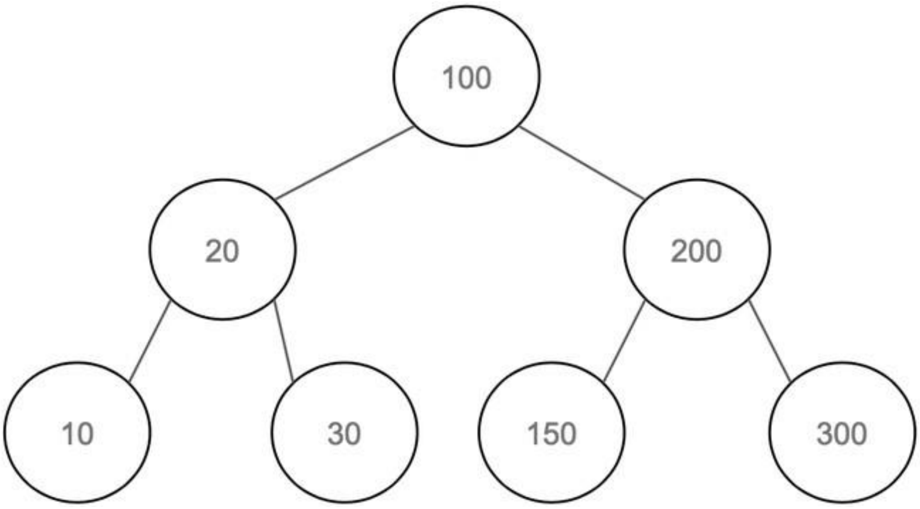
Algorithm: Postorder Traversal of BST

1. Start at the root node
2. Recursively traverse the left subtree
3. Recursively traverse the right subtree
4. Visit the root node and process it

Algorithm: Inorder Traversal of BST

1. Start at the root node
2. Recursively traverse the left subtree
3. Visit the root node and process it
4. Recursively traverse the right subtree

An example BST :



Preorder Traversal:
100, 20, 10, 30, 200, 150, 300

Postorder Traversal:
10, 30, 20, 150, 300, 200, 100

Inorder Traversal:
10, 20, 30, 100, 150, 200, 300

Algorithm for Implementation of BST:

1. Node Structure Definition

- Define a structure TreeNode with data (int), leftChild (pointer to left subtree), rightChild (pointer to right subtree)

2. Create Node

- Allocate memory for a new node.
- Set data to value and both children to NULL.
- Return the new node pointer.

3. Insert Node

- If root is NULL, create and return a new node
- If value < root->data, recursively insert in the left subtree
- If value > root->data, recursively insert in the right subtree
- Return the root

4. Search Node

- If root is NULL, return false else root->data == value, return true
- If value < root->data, search in the left subtree
- If value > root->data, search in the right subtree

5. Inorder Traversal

- If root is NOT NULL:
  1. Traverse the left child.
  2. Print root->data.
  3. Traverse the right child.

6. Main Function

- Initialize root = NULL and use a menu to perform insert, search, and traversal operations

Implementation Details:

1) Enlist all the Steps followed and various options explored.

1. **Define the TreeNode Structure**:
   - Created a structure TreeNode with three members: data, leftChild, and rightChild
2. **Node Creation**:
   - Implemented the createNode function to allocate memory for a new node and initialize its data and child pointers
3. **Insertion Functionality**:
   - Developed the insertNode function to insert a value into the BST. It recursively finds the correct position based on value comparisons
4. **Search Functionality**:
   - Implemented the searchNode function to find a value in the tree. It traverses left or right based on comparisons and outputs whether the value was found
5. **Traversal Functions**:
   - Created three traversal functions:

     <u>Inorder Traversal:</u> Visits left child, root, then right child.

     <u>Preorder Traversal:</u> Visits root, left child, then right child.

     <u>Postorder Traversal:</u> Visits left child, right child, then root.
6. **Menu System**:
   - Designed a loop in main to display a menu for user interaction, allowing them to choose various operations like inserting, searching, or traversing the tree
7. **User Input Handling**:
   - Used scanf to accept user input for different operations, ensuring each choice is processed accordingly

Assumptions made for Input:

1. **Valid Integer Input**: It is assumed that users will input valid integers when prompted.

2. **No Duplicate Values**: The program does not handle duplicate values. It assumes that each value inserted will be unique.

3. **Continuous Operation**: The program assumes the user will continuously choose options until they decide to exit.

Built-In Functions Used:

1. malloc( ): Allocates memory for a new node in the tree

2. printf( ): Displays output to the console

3. scanf( ) : Reads user input from the console

4. exit( ): Terminates the program.

Program source code for Implementation of BST & Binary tree traversal techniques :

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

typedef struct {
    Node* root;
} BST;

BST* create() {
    BST* bst = (BST*)malloc(sizeof(BST));
    bst->root = NULL;
    return bst;
}

BST* insert(BST* bst, int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    if (bst->root == NULL) {
        bst->root = node;
    } else {
        Node* curr = bst->root;
        Node* parent = NULL;

        while(curr != NULL) {
            parent = curr;
            if (data < curr->data) {
                curr = curr->left;
            } else {
                curr = curr->right;
            }
        }
        if (data < parent->data) {
            parent->left = node;
        } else {
```

```
            parent->right = node;
        }
    }
    return bst;
}

void search(BST* bst, int data) {
    Node* curr = bst->root;
    while (curr != NULL) {
        if (data == curr->data) {
            printf("Found %d in the BST\n", data);
            return;
        } else if (data < curr->data) {
            curr = curr->left;
        } else {
            curr = curr->right;
        }
    }
    printf("%d not found in the BST\n", data);
}

void inorder(Node* node) {
    if (node != NULL) {
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}

void preorder(Node* node) {
    if (node != NULL) {
        printf("%d ", node->data);
        preorder(node->left);
        preorder(node->right);
    }
}

void postorder(Node* node) {
    if (node != NULL) {
        postorder(node->left);
        postorder(node->right);
        printf("%d ", node->data);
    }
}
```

```
int main() {
    BST* bst = create();

    int choice, data;
    while (1) {
        printf("1. Insert node\n");
        printf("2. Search for node\n");
        printf("3. Perform inorder traversal\n");
        printf("4. Perform preorder traversal\n");
        printf("5. Perform postorder traversal\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &data);
                bst = insert(bst, data);
                break;
            case 2:
                printf("Enter the value to search for: ");
                scanf("%d", &data);
                search(bst, data);
                break;
            case 3:
                printf("Inorder: ");
                inorder(bst->root);
                printf("\n");
                break;
            case 4:
                printf("Preorder: ");
                preorder(bst->root);
                printf("\n");
                break;
            case 5:
                printf("Postorder: ");
                postorder(bst->root);
                printf("\n");
                break;
            case 6:
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
```

```
    }
    return 0;
}
```

Output Screenshots for Each Operation:

```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs\" ; if ($
) { gcc BST.c -o BST } ; if ($?) { .\BST }
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 2
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 2
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 4
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 5
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
```

```
Enter your choice: 1
Enter the value to insert: 7
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 2
Enter the value to search for: 3
3 not found in the BST
```

```
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 3
Inorder: 2 2 4 5 7
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 4
Preorder: 2 2 4 5 7
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 5
Postorder: 7 5 4 2 2
1. Insert node
2. Search for node
3. Perform inorder traversal
4. Perform preorder traversal
5. Perform postorder traversal
6. Exit
Enter your choice: 6
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\DS\Programs>
```

Conclusion:-

The above program highlights the implementation of a Binary Search Tree in C with functions of search and tree traversals.

PostLab Questions:

1) Write an ADT for tree data structure

Value Definition
Abstract typedef TreeType <ElementType>

Condition: None

Operator Definition

- Abstract TreeType create <>

  Precondition: None
  Postcondition: Set root node to NULL

- Abstract TreeType insert <ElementType el>

  Precondition: A tree has been created
  Postcondition: el inserted in the correct position

- Abstract Boolean search <ElementType el>

  Precondition: A tree has been created
  Postcondition: Returns true if the element el is found in the tree, otherwise returns false

- Abstract void Inorder < >

  Precondition: A tree has been created and contains at least one element
  Postcondition: Returns a list of the tree's elements in inorder (left subtree, root, right subtree)

- Abstract void Preorder < >

  Precondition: A tree has been created and contains at least one element
  Postcondition: Returns a list of the tree's elements in preorder (root, left subtree, right subtree)

- Abstract void Postorder < >

  Precondition: A tree has been created and contains at least one element
  Postcondition: Returns a list of the tree's elements in postorder (left subtree, right subtree, root)

2) Write a program to count the nodes in the Binary tree

Code-

```
#include <stdio.h>
#include <stdlib.h>


struct TreeNode {
    int data;
    struct TreeNode* leftChild;
    struct TreeNode* rightChild;
};

typedef struct TreeNode TreeNode;


TreeNode* createNode(int value)
{
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = value;
    newNode->leftChild = NULL;
    newNode->rightChild = NULL;
    return newNode;
}


TreeNode* insertNode(TreeNode* root, int value)
{
    if (root == NULL)
    {
        return createNode(value);
    }
    if (value < root->data)
    {
        root->leftChild = insertNode(root->leftChild, value);
    }
    else if (value > root->data)
    {
        root->rightChild = insertNode(root->rightChild, value);
    }
    return root;
}


int countNodes(TreeNode* root)
{
    if (root == NULL)
```

```
    {
        return 0;
    }
    return 1 + countNodes(root->leftChild) + countNodes(root->rightChild);
}


int main()
{
    TreeNode* root = NULL;
    int choice, value;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Insert Node\n");
        printf("2. Count Nodes\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("The total number of nodes in the tree: %d\n",
countNodes(root));
                break;
            case 3:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output-

```
Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 1
Enter value to insert: 44

Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 1
Enter value to insert: 16

Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 1
Enter value to insert: 55

Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 1
Enter value to insert: 3

Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 2
The total number of nodes in the tree: 4

Menu:
1. Insert Node
2. Count Nodes
3. Exit
Enter your choice: 3
Exiting...
```

3) Write a program to find the height of the Binary tree.

Code-

```
#include <stdio.h>
#include <stdlib.h>


struct TreeNode {
    int data;
    struct TreeNode* leftChild;
    struct TreeNode* rightChild;
};

typedef struct TreeNode TreeNode;

TreeNode* createNode(int value)
{
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = value;
    newNode->leftChild = NULL;
    newNode->rightChild = NULL;
    return newNode;
}


TreeNode* insertNode(TreeNode* root, int value)
{
    if (root == NULL)
    {
        return createNode(value);
    }
    if (value < root->data)
    {
        root->leftChild = insertNode(root->leftChild, value);
    }
    else if (value > root->data)
    {
        root->rightChild = insertNode(root->rightChild, value);
    }
    return root;
}

int findHeight(TreeNode* root)
{
    if (root == NULL)
    {
```

```
        return -1;
    }
    int leftHeight = findHeight(root->leftChild);
    int rightHeight = findHeight(root->rightChild);
    return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);
}


int main()
{
    TreeNode* root = NULL;
    int choice, value;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Insert Node\n");
        printf("2. Find Height of the Tree\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("The height of the tree is: %d\n", findHeight(root));
                break;
            case 3:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output-

```
Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 33

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 1

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 55

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 16

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 3

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 1
Enter value to insert: 81

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 2
The height of the tree is: 4

Menu:
1. Insert Node
2. Find Height of the Tree
3. Exit
Enter your choice: 3
Exiting...
```

4) **The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Construct the Binary Search Tree and perform the Postorder Traversal for the same.**