Neh Samir Joshi

19BEC1112

CSE2003

Data Structures and Algorithms

[LAB]

LAB – 7

Binary Trees and Binary Search Trees

Aim: To implement binary trees and binary search trees in C and learn about the different types of traversals.

Software Required: Code editor (e.g. VS Code, Dev C++), GCC/G++ compiler

Task 1: To create a binary tree and perform basic functions such as insertion, deletion and traversal.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data_element;
    struct node *left, *right;
};

struct node *new_node(int data_element)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data_element = data_element;
    temp->left = temp->right = NULL;
    return temp;
}

void display(struct node *root)
{
```

```
    if (root != NULL)
    {
        display(root->left);
        printf("%d \n", root->data_element);
        display(root->right);
    }
}

struct node *insert(struct node *node, int data_element)
{

    if (node == NULL)
        return new_node(data_element);
    if (data_element < node->data_element)
    {
        node->left = insert(node->left, data_element);
    }
    else if (data_element > node->data_element)
    {
        node->right = insert(node->right, data_element);
    }
    return node;
}

int main()
{

    struct node *root = NULL;
    root = insert(root, 10);
    insert(root, 20);
    insert(root, 30);
    insert(root, 40);
    insert(root, 50);
    insert(root, 60);
    insert(root, 70);

    display(root); // Function to display the binary tree elements

    return 0;
}
```

```
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code> cd "c:\Users\OS\Desktop\6th Semeste
r\CSE2003 Data Structures and Algorithms\Code\LAB6\" ; if ($?) { gcc binary_tree.c -o binary_tree } ; if ($?) { .\bi
nary_tree }
10
20
30
40
50
60
70
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB6>
```

Task 2: To implement a binary search tree and perform insertion, deletion and traversal.

Code:

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *left;
    Node *right;
};
struct Node* GetNewNode(int data){
    Node *newNode = new Node;
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
void Preorder(struct Node *root){
    if (root==NULL) return;
    cout << root->data << endl;
    Preorder(root->left);
    Preorder(root->right);
}
void Inorder(struct Node *root){
    if (root==NULL) return;
    Inorder(root->left);
    cout << root->data << endl;
    Inorder(root->right);
}
void Postorder(struct Node *root){
    if (root==NULL) return;
    Postorder(root->left);
```

```
    Postorder(root->right);
    cout << root->data << endl;
}
struct Node* Insert(struct Node *root, int data){
    if (root == NULL){
        root = GetNewNode(data);
    }
    else if (data <= root->data){
        root->left = Insert(root->left, data);
    }
    else {
        root->right = Insert(root->right, data);
    }
    return root;
}
bool Search(struct Node* root, int value){
    if (root==NULL) return false;
    else if(root->data==value) return true;
    else if(value <= root->data) return Search(root->left, value);
    else return Search(root->right, value);
}
struct Node *FindMin(struct Node *root){
    struct Node *current = root;
    while (current->left != NULL){
        current = current->left;
    }
    return current;
}
struct Node* Delete(struct Node *root, int data){
    if (root==NULL) return root;
    else if(data < root->data) root->left = Delete(root->left, data);
    else if (data > root->data) root->right = Delete(root->right, data);
    else {
        if (root->left == NULL && root->right == NULL){ //Case if target node has
no childrrn
            root = NULL;
        }
        else if (root->left == NULL){   //These two cases are for when the target
node has 1 child
            struct Node *temp = root;
            root = root->right;
            delete temp;
        }
        else if (root->right == NULL){
            struct Node *temp = root;
```

```
            root = root->left;
            temp = NULL;
        }
        else {  //When target node has two children
            struct Node *temp = FindMin(root->right);
            root->data = temp->data;
            root->right = Delete(root->right, temp->data);
        }
        return root;
    }
    return root;

}

int main(){

    struct Node *root = new Node;
    root = NULL;
    root = Insert(root, 12);
    root = Insert(root, 5);
    root = Insert(root, 15);
    root = Insert(root, 3);
    root = Insert(root, 7);
    root = Insert(root, 1);
    root = Insert(root, 9);

    cout << "Visualizing the BST: Inorder" << endl;
    Inorder(root);

    cout << "Visualizing the BST: Preorder" << endl;
    Inorder(root);

    cout << "Visualizing the BST: Postorder" << endl;
    Inorder(root);

    Delete(root, 1);
    cout << "After deleting 1" << endl;
    Inorder(root);
    return 0;
}
```

Output:

```
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB6> cd "c:\Users\OS\Desktop\6th Se
mester\CSE2003 Data Structures and Algorithms\Code\LAB6\" ; if ($?) { g++ bst_basic.cpp -o bst_basic } ; if ($?) { .
\bst_basic }
Visualizing the BST: Inorder
1
3
5
7
9
12
15
Visualizing the BST: Preorder
1
3
5
7
9
12
15
Visualizing the BST: Postorder
1
3
5
7
9
12
15
After deleting 1
3
5
7
9
12
15
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB6>
```

Conclusion

Thus, we have successfully implemented binary trees and binary search trees in C. We have also performed basic functions such as insertion and deletion and looked at the three different types of traversals. Hence, the experiment is complete.

Neh Samir Joshi

19BEC1112