# Data Structures Odyssey: Exploring the Foundations of Computing

| | | |
|---|---|---|
| **Ex. No.:09** | **Implementation of Binary Search tree** | **Date:25/04/2024** |

**Write a C program to implement a Binary Search Tree and perform the following operations.**

**(i)** **Insert**

**(ii)** **Delete**

**(iii)** **Search**

**(iv)** **Display**

**Algorithm:**

1) Start
2) Define a Node structure with data, left child pointer, and right child pointer.
3) Initialize a root pointer to NULL.
4) Create functions for the following operations: a. Insert:

- If the tree is empty, create a new Node and set it as the root.
- Otherwise, traverse the tree starting from the root:
- If the data is less than the current Node's data, move to the left child.
- If the data is greater than the current Node's data, move to the right child. - Repeat until reaching a NULL child pointer, then insert the new Node. b. Search:
- Start from the root and compare the data with each Node:
- If the data matches, return the Node.
- If the data is less than the current Node's data, move to the left child.
- If the data is greater than the current Node's data, move to the right child.
- Repeat until finding the data or reaching a NULL child pointer.

5) Test the operations by inserting elements into the tree and searching for specific values. 6) Stop

PROGRAM;

```
#include <stdio.h>
#include <stdlib.h> struct
BinaryTreeNode { int
key;
struct BinaryTreeNode *left, *right;
};

struct BinaryTreeNode* newNodeCreate(int value)
{
struct BinaryTreeNode* temp =
(struct BinaryTreeNode*)malloc(
sizeof(struct BinaryTreeNode));
temp->key = value; temp->left =
temp->right = NULL; return temp;
}
struct BinaryTreeNode*
searchNode(struct BinaryTreeNode* root, int target)
{
if (root == NULL || root->key == target) { return
root;
}
if (root->key < target) { return
searchNode(root->right, target);
}
return searchNode(root->left, target);
}

struct BinaryTreeNode*
insertNode(struct BinaryTreeNode* node, int value)
```

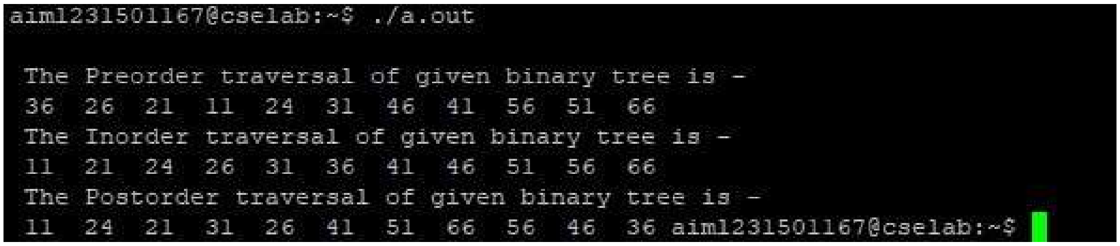
```
{
if (node == NULL) { return
newNodeCreate(value);
}
if (value < node->key) { node->left =
insertNode(node->left, value);
}
else if (value > node->key) { node->right =
insertNode(node->right, value);
}
return node;
}

void postOrder(struct BinaryTreeNode* root)
{
if (root != NULL) {
postOrder(root->left);
postOrder(root->right); printf("
%d ", root->key);
}
}

void inOrder(struct BinaryTreeNode* root)
{
if (root != NULL) {
inOrder(root->left); printf("
%d ", root->key);
inOrder(root->right);
}
}
void preOrder(struct BinaryTreeNode* root)
{
if (root != NULL) { printf("
%d ", root->key);
```

```
preOrder(root->left);
preOrder(root->right);
}
}

struct BinaryTreeNode* findMin(struct BinaryTreeNode* root)
{
if (root == NULL) { return
NULL;
}
else if (root->left != NULL) { return
findMin(root->left);
}
return root;
}

struct BinaryTreeNode* delete (struct BinaryTreeNode* root, int
x)
{
if (root == NULL) return
NULL;

if (x > root->key) { root->right =
delete (root->right, x);
}
else if (x < root->key) { root->left
= delete (root->left, x);
}
else {
if (root->left == NULL && root->right == NULL) {
free(root); return NULL;
}
```

```
else if (root->left == NULL ||
root->right == NULL) { struct
BinaryTreeNode* temp; if
(root->left == NULL) { temp =
root->right;
} else { temp =
root->left;
} free(root);
return temp;
} else
{
struct BinaryTreeNode* temp = findMin(root->right);
root->key = temp->key; root->right = delete (root-
>right, temp->key);
}
}
return root;
}

int main()
{
struct BinaryTreeNode* root = NULL;

root = insertNode(root, 50);
insertNode(root, 30);
insertNode(root, 20);
insertNode(root, 40);
insertNode(root, 70);
insertNode(root, 60);
insertNode(root, 80); if
(searchNode(root, 60) != NULL) {
printf("60 found");
```

```
} else { printf("60 not
found");
}

printf("\n"); postOrder(root);
printf("\n");

preOrder(root);
printf("\n"); inOrder(root);
printf("\n");

struct BinaryTreeNode* temp = delete (root, 70);
printf("After Delete: \n"); inOrder(root);

return 0;
}
```

OUTPUT:



```
aiml231501167@cselab:~$ ./a.out

 The Preorder traversal of given binary tree is -
 36  26  21  11  24  31  46  41  56  51  66
 The Inorder traversal of given binary tree is -
 11  21  24  26  31  36  41  46  51  56  66
 The Postorder traversal of given binary tree is -
 11  24  21  31  26  41  51  66  56  46  36 aiml231501167@cselab:~$
```

**RESULT: Thus, the program was successfully executed.**