



Experiment No.9
Implement Binary Search Tree ADT using Linked List.
Name: Sakshi Patil
Roll No:44
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 9: Binary Search Tree Operations Aim
: Implementation of Binary Search Tree ADT using Linked List.

Objective:

- 1) Understand how to implement a BST using a predefined BST ADT.
- 2) Understand the method of counting the number of nodes of a binary tree.

Theory:

A binary tree is a finite set of elements that is either empty or partitioned into disjoint subsets. In other words nodes in a binary tree have at most two children and each child node is referred to as left or right child.

Traversals in trees can be in one of the three ways: preorder, postorder, inorder.

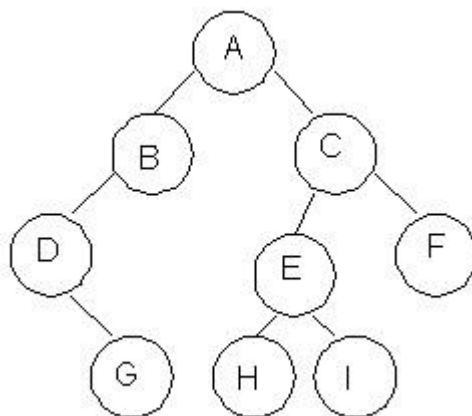
Preorder Traversal

Here the following strategy is followed in sequence

1. Visit the root node R
2. Traverse the left subtree of R
3. Traverse the right subtree of R



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science



Description	Output
Visit Root	A
Traverse left sub tree – step to B then D	ABD
Traverse right subtree – step to G	ABDG
As left subtree is over. Visit root , which is already visited so go for right subtree	ABDGC
Traverse the left subtree	ABDGCEH
Traverse the right sub tree	ABDGCEHIF

Inorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Visit the root node R
3. Traverse the right sub tree of R

Description	Output
Start with root and traverse left sub tree from A-B-D	D
As D doesn't have left child visit D and go for right subtree of D which is G so visit this.	DG
Backtrack to D and then to B and visit it.	DGB
Backtrack to A and visit it	DGBA
Start with right sub tree from C-E-H and visit H	DGBAH
Now traverse through parent of H which is E and then I	DGBAHEI



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Backtrack to C and visit it and then right subtree of E which is F	DGBAHEICF
--	-----------

Postorder Traversal

Here the following strategy is followed in sequence

1. Traverse the left subtree of R
2. Traverse the right sub tree of R
3. Visit the root node R

Description	Output
Start with left sub tree from A-B-D and then traverse right sub tree to get G	G
Now Backtrack to D and visit it then to B and visit it.	GD
Now as the left sub tree is over go for right sub tree	GDB
In right sub tree start with leftmost child to visit H followed by I	GDBHI
Visit its root as E and then go for right sibling of C as F	GDBHIEF
Traverse its root as C	GDBHIEFC
Finally a root of tree as A	GDBHIEFCA

Algorithm

Algorithm: PREORDER(ROOT)

Algorithm :

Function Pre-order(root)

- Start
- If root is not null then

Display the data in root

Call pre order with left pointer of root(root -> left)

Call pre order with right pointer of root(root -> right)

- Stop

Algorithm: INORDER(ROOT)

Algorithm :



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Function in-order(root)

- Start
- If root is not null then

Call in order with left pointer of root (root -> left)

Display the data in root

Call in order with right pointer of root(root -> right)

- Stop

Algorithm: POSTORDER(ROOT)

Algorithm :

Function post-order (root)

- Start
- If root is not null then

Call post order with left pointer of root (root -> left)

Call post order with right pointer of root (root -> right)

Display the data in root

- Stop

Code:

```
#include<stdio.h>

#include<conio.h> #include<malloc.h> struct node { int
data; struct node *left; struct node *right; }; struct node
*tree; void create_tree(struct node *); struct node
*insertElement(struct node *, int); void
preorderTraversal(struct node *); void
inorderTraversal(struct node *); void
postorderTraversal(struct node *); struct node
*findSmallestElement(struct node *); struct node
*findLargestElement(struct node *);
```



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

```
struct node *deleteElement(struct node *, int);

struct node *mirrorImage(struct node *); int

totalNodes(struct node *); int

totalExternalNodes(struct node *); int

totalInternalNodes(struct node *); int

Height(struct node *); struct node

*deleteTree(struct node *); int main() { int

option, val; struct node *ptr; create_tree(tree);

clrscr();

do { printf("\n *****MAIN MENU***** \n");

printf("\n 1. Insert Element"); printf("\n 2. Preorder

Traversal"); printf("\n 3. Inorder Traversal"); printf("\n

4. Postorder Traversal"); printf("\n 5. Find the smallest

element"); printf("\n 6. Find the largest element");

printf("\n 7. Delete an element"); printf("\n 8. Count the

total number of nodes"); printf("\n 9. Count the total

number of external nodes"); printf("\n 10. Count the

total number of internal nodes"); printf("\n 11.

Determine the height of the tree"); printf("\n 12. Find

the mirror image of the tree"); printf("\n 13. Delete the

tree"); printf("\n 14. Exit");

printf("\n\n Enter your option : "); scanf("%d", &option);

switch(option) { case 1: printf("\n Enter the value of the new

node : "); scanf("%d", &val); tree = insertElement(tree,

val); break; case 2: printf("\n The elements of the tree are :

\n"); preorderTraversal(tree); break; case 3: printf("\n The
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
elements of the tree are : \n"); inorderTraversal(tree); break;

case 4: printf("\n The elements of the tree are : \n");

postorderTraversal(tree); break; case 5: ptr =

findSmallestElement(tree); printf("\n Smallest element is

:%d",ptr->data); break; case 6: ptr =

findLargestElement(tree); printf("\n Largest element is :

%d", ptr->data); break; case 7: printf("\n Enter the element

to be deleted : "); scanf("%d", &val); tree =

deleteElement(tree, val); break; case 8: printf("\n Total no.

of nodes = %d", totalNodes(tree)); break; case 9: printf("\n

Total no. of external nodes = %d", totalExternalNodes(tree));

break; case 10: printf("\n Total no. of internal nodes = %d",

totalInternalNodes(tree)); break; case 11: printf("\n The

height of the tree = %d",Height(tree)); break; case 12:tree =

mirrorImage(tree); break; case 13: tree = deleteTree(tree);

break;

}

}while(option!=14);

getch(); return 0;

} void create_tree(struct node *tree) { tree

= NULL;

} struct node *insertElement(struct node *tree, int val) {

struct node *ptr, *nodeptr, *parentptr; ptr = (struct

node*)malloc(sizeof(struct node)); ptr->data = val;

ptr->left = NULL; ptr->right = NULL;
```



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

```
if(tree==NULL) { tree=ptr; tree->left=NULL; tree->right=NULL;
```

```
} else { parentptr=NULL;
```

```
nodeptr=tree;
```

```
while(nodeptr!=NULL) {
```

```
parentptr=nodeptr; if(valdata)
```

```
nodeptr=nodeptr->left; else
```

```
nodeptr = nodeptr->right; }
```

```
if(valdata) parentptr->left = ptr;
```

```
else parentptr->right = ptr;
```

```
} return tree;
```

```
} void preorderTraversal(struct node *tree)
```

```
if(tree != NULL) { printf("%d\t", tree->
```

```
data); preorderTraversal(tree->left);
```

```
preorderTraversal(tree->right);
```

```
}
```

```
} void inorderTraversal(struct node *tree)
```

```
{ if(tree != NULL) {
```

```
inorderTraversal(tree->left);
```

```
printf("%d\t", tree->data);
```

```
inorderTraversal(tree->right);
```

```
}
```

```
} void postorderTraversal(struct node
```

```
*tree) { if(tree != NULL) {
```

```
postorderTraversal(tree->left);
```



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

```
postorderTraversal(tree->right);

printf("%d\t", tree->data);

} struct node *findSmallestElement(struct node
*tree) { if (tree == NULL) || (tree->left == NULL))
return tree; else return findSmallestElement(tree -
>left);

} struct node *findLargestElement(struct node *tree)
{ if( (tree == NULL) || (tree->right == NULL)) return
tree; else return findLargestElement(tree->right);

} struct node *deleteElement(struct node *tree, int val) {
struct node *cur, *parent, *suc, *psuc, *ptr; if(tree->
left==NULL) { printf("\n The tree is empty ");
return(tree); } parent = tree; cur = tree->left;
while(cur!=NULL && val!= cur->data) { parent = cur; cur
= (valdata)? cur->left:cur->right; } if(cur == NULL) {
printf("\n The value to be deleted is not present in the tree");
return(tree);

} if(cur->left == NULL) ptr = cur->right;
else

if(cur->right == NULL)

ptr = cur->left; else {

// Find the in-order successor and its parent

psuc = cur; cur = cur->left; while(suc->
left!=NULL) {

psuc = suc; suc = suc->
left; } if(cur==psuc) {
```




Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

```
// Situation 1  suc->left
= cur->right;

} else {

// Situation 2  suc->left
= cur->left;  psuc->left
=  suc->right;    suc-
>right = cur->right;  }

ptr = suc;

} // Attach ptr to the parent node
if(parent->left == cur) parent-
>left=ptr;      else  parent-
>right=ptr; free(cur); return tree;
} int totalNodes(struct node *tree)
{ if(tree==NULL) return 0; else return(totalNodes(tree-
>left) + totalNodes(tree->right) + 1);

}  int  totalExternalNodes(struct  node  *tree)  {
if(tree==NULL) return 0; else if((tree->left==NULL)
&& (tree->right==NULL))
return 1; else return (totalExternalNodes(tree->left) +
totalExternalNodes(tree->right));

} int totalInternalNodes(struct node *tree) { if( (tree==NULL) ||
((tree->left==NULL) && (tree->right==NULL)))
return 0; else return (totalInternalNodes(tree->left) + totalInternalNodes(tree-
>right) + 1); } int Height(struct node *tree)
{ int leftheight, rightheight; if(tree==NULL)
return 0; else
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
{ leftheight = Height(tree->left);  
rightheight = Height(tree->right);  
if(leftheight > rightheight) return  
(leftheight + 1); else  
return (rightheight + 1);  
}  
  
} struct node *mirrorImage(struct node  
*tree) { struct node *ptr; if(tree!=NULL) {  
mirrorImage(tree->left); mirrorImage(tree->  
>right); ptr=tree->left; ptr->left = ptr->  
>right; tree->right = ptr;  
}  
}  
  
} struct node *deleteTree(struct node *tree) {  
if(tree!=NULL) { deleteTree(tree->left);  
deleteTree(tree->right);  
free(tree);  
}  
}
```

Output:



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
*****MAIN  MENU*****
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Count the total number of external nodes
10. Count the total number of internal nodes
11. Determine the height of the tree
12. Find the mirror image of the tree
13. Delete the tree
14. Exit
Enter your option : 1
Enter the value of the new node : 1
Enter the value of the new node : 2
Enter the value of the new node : 4
Enter your option : 3
2 1 4
Enter your option : 14
```

Conclusion:

Write a function in C program to count the number of nodes in a binary search tree?

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

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

```
// Definition for a BST node
```

```
struct Node {    int data;
```

```
struct Node* left;    struct
```

```
Node* right;
```

```
};
```

```
// Function to create a new BST node struct
```

```
Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));    newNode->data = data;
```

```
    newNode->left = newNode->right = NULL;
```

```
    return newNode;
```

```
}
```



Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

```
// Function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL)    return createNode(data);
    if (data < root->data)    root->left =
    insert(root->left, data);    else if (data > root-
    >data)    root->right = insert(root->right,
    data);    return root;
}

// Function to count the number of nodes in the BST
int countNodes(struct Node* root) {    if (root ==
    NULL)    return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}

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

    // Insert elements into the BST
    root = insert(root, 10);    root =
    insert(root, 5);    root =
    insert(root, 15);    root =
    insert(root, 3);    root =
    insert(root, 7);    root =
    insert(root, 12);    root =
    insert(root, 18);

    // Count the number of nodes in the BST
    int nodeCount = countNodes(root);    printf("Number of
    nodes in the BST: %d\n", nodeCount);

    return 0;
}
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science