**Batch: A2 Roll No.: 16010123032**

**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

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. <https://www.geeksforgeeks.org/binary-tree-data-structure/>
3. <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**

**Preorder Traversal Algorithm:**

1. Visit the root node:
   * Print the value of the root node.
2. Traverse the left subtree:
   * Recursively call the preorder traversal function on the left child of the root node.
3. Traverse the right subtree:
   * Recursively call the preorder traversal function on the right child of the root node.

**Algorithm: Postorder Traversal of BST**

**Postorder Traversal Algorithm:**

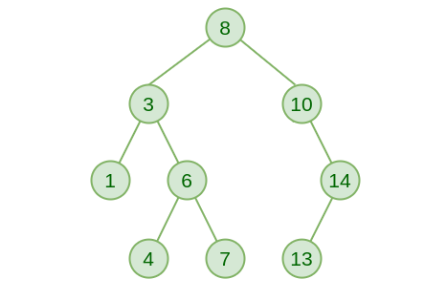
1. Traverse the left subtree:
   * Recursively call the postorder traversal function on the left child of the root node.
2. Traverse the right subtree:
   * Recursively call the postorder traversal function on the right child of the root node.
3. Visit the root node:
   * Print the value of the root node.

**Algorithm: Inorder Traversal of BST**

**Inorder Traversal Algorithm:**

1. Traverse the left subtree:
   * Recursively call the inorder traversal function on the left child of the root node.
2. Visit the root node:
   * Print the value of the root node.
3. Traverse the right subtree:
   * Recursively call the inorder traversal function on the right child of the root node.

**An example BST:**

****

**Preorder Traversal:**

In preorder traversal, the root node is visited first, followed by a recursive traversal of the left subtree and then the right subtree. This results in a depth-first traversal where the root node is always visited before its children.

**Eg: 8 3 1 6 4 7 10 14 13**

**Postorder Traversal:**

In postorder traversal, the left subtree is visited first, then the right subtree, and finally the root node. This traversal is also depth-first, but the root node is visited after all its descendants**.**

**Eg: 1 4 7 6 3 13 14 10 8**

**Inorder Traversal:**

In inorder traversal, the left subtree is visited first, then the root node, and finally the right subtree. This traversal results in a sorted order of the nodes' values when the BST is balanced. It is commonly used to visit nodes in ascending order of their keys.

**Eg: 1 3 4 6 7 8 10 13 14**

**Algorithm for Implementation of BST:**

**1. Create Node (Implemented in Create function):**

* This function takes an integer value as input.
* It allocates memory for a new Node using malloc.
* It assigns the input value to the key field of the new node.
* It initializes both left and right pointers of the new node to NULL (indicating empty child nodes).
* It returns the pointer to the newly created node.

**2. Search (Implemented in search function):**

* This function takes a root node of the BST and a target value as input.
* It checks if the root node is NULL or if its key matches the target value.
  + If NULL, the target is not found and the function returns NULL.
  + If the key matches the target, the function returns the root node (target found).
* If the target value is less than the key of the root node, the function recursively searches the right subtree by calling search(root->right, target).
* Otherwise, the function recursively searches the left subtree by calling search(root->left, target).

**3. Insert (Implemented in insert function):**

* This function takes a root node of the BST and a value to insert as input.
* It checks if the root node is NULL.
  + If NULL, a new node is created with the given value and returned (base case for an empty tree).
* Otherwise, it compares the value to insert with the key of the root node.
  + If the value is less than the key, the function recursively inserts it into the left subtree by calling insert(node->left, value).
  + If the value is greater than the key, the function recursively inserts it into the right subtree by calling insert(node->right, value).
* The original root node is returned (no changes needed if the value already exists in the BST due to the BST property).

**4. Inorder Traversal (Implemented in inorder function):**

* This function takes a root node of the BST as input.
* It checks if the root node is NULL.
  + If NULL, the function does nothing (base case for an empty subtree).
* Otherwise, it performs an in-order traversal:
  + It recursively calls inorder(node->left) to traverse the left subtree.
  + It prints the key of the current node.
  + It recursively calls inorder(node->right) to traverse the right subtree.

**Implementation Details:**

**Enlisted below are all the Steps followed and the Algorithm is given above.**

**Assumptions made for Input:**

**A Non-negative Integer value shall be entered by user.**

**Built-In Functions Used:**

**None**

**Program source code for Implementation of BST & Binary tree traversal techniques:**#include <stdio.h>

#include <stdlib.h>

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

};

struct Node\* Create(int value)

{

struct Node\* temp=(struct Node\*)malloc(sizeof(struct Node));

temp->key = value;

temp->left = NULL;

temp->right = NULL;

return temp;

}

struct Node\* search(struct Node\* root, int target)

{

if (root == NULL || root->key == target)

{

return root;

}

if (root->key < target)

{

return search(root->right, target);

}

return search(root->left, target);

}

struct Node\* insert(struct Node\* node, int value)

{

if (node == NULL)

{

return Create(value);

}

if (value < node->key)

{

node->left = insert(node->left, value);

}

else if (value > node->key)

{

node->right = insert(node->right, value);

}

return node;

}

void inorder(struct Node\* node)

{

if (node != NULL)

{

inorder(node->left);

printf("%d ", node->key);

inorder(node->right);

}

}

int main()

{

struct BinaryTreeNode\* root = NULL;

int t=1;

while(t==1)

{

int ch,n;

printf("1.Insert 2.Search 3.InOrder 4.Exit \n");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(root == NULL)

{

printf("Enter value: ");

scanf("%d",&n);

root = insert(root,n);

}

else

{

printf("Enter value: ");

scanf("%d",&n);

insert(root,n);

}

break;

case 2:

printf("Enter value to search for: ");

scanf("%d",&n);

if(search(root,n) != NULL)

{

printf("%d is found \n",n);

}

else

{

printf("%d is not found \n",n);

}

break;

case 3:

inorder(root);

printf("\n");

break;

case 4:

t=0;

printf("Exiting............ \n");

break;

default:

printf("Invalid choice \n");

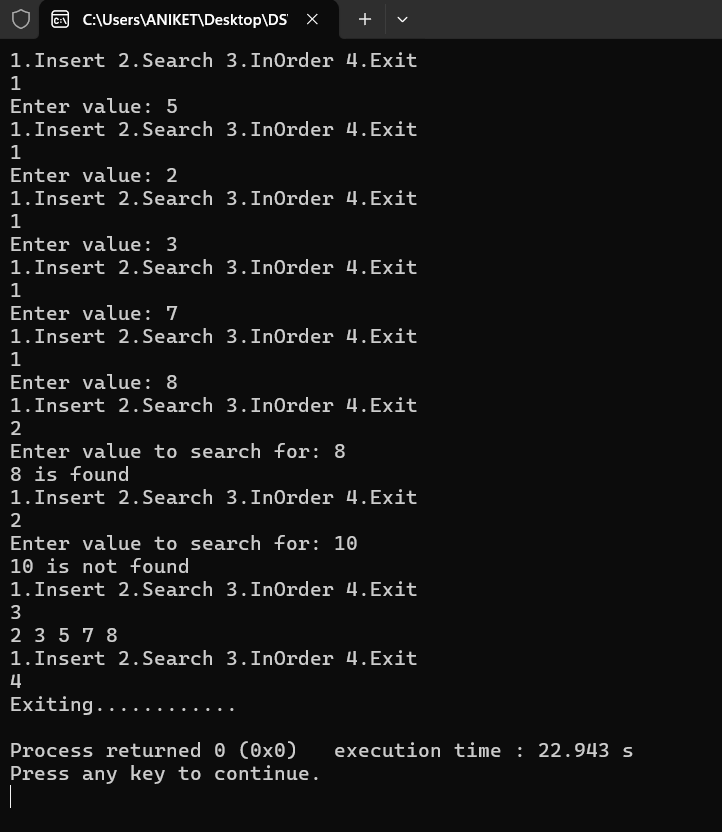
break;

}

}

}

**Output Screenshots for Each Operation:**

****

**Conclusion:-**

**We successfully implemented BST in C.**

**PostLab Questions:**

1. **Write a program for inorder, Preorder and Postorder traversal without using recursion.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int key;

struct Node\* left;

struct Node\* right;

};

struct Node\* newNode(int key) {

struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

temp->key = key;

temp->left = temp->right = NULL;

return temp;

}

struct Stack {

struct Node\* data;

struct Stack\* next;

};

struct Stack\* newStackNode(struct Node\* node) {

struct Stack\* stackNode = (struct Stack\*)malloc(sizeof(struct Stack));

stackNode->data = node;

stackNode->next = NULL;

return stackNode;

}

void push(struct Stack\*\* top, struct Node\* node) {

struct Stack\* stackNode = newStackNode(node);

stackNode->next = \*top;

\*top = stackNode;

}

int isEmpty(struct Stack\* top) {

return !top;

}

struct Node\* pop(struct Stack\*\* top) {

if (isEmpty(\*top)) {

return NULL;

}

struct Stack\* temp = \*top;

\*top = (\*top)->next;

struct Node\* poppedNode = temp->data;

free(temp);

return poppedNode;

}

void inorderTraversal(struct Node\* root) {

struct Stack\* stack = NULL;

struct Node\* current = root;

while (current != NULL || !isEmpty(stack)) {

while (current != NULL) {

push(&stack, current);

current = current->left;

}

current = pop(&stack);

printf("%d ", current->key);

current = current->right;

}

}

void preorderTraversal(struct Node\* root) {

if (root == NULL) return;

struct Stack\* stack = NULL;

push(&stack, root);

while (!isEmpty(stack)) {

struct Node\* node = pop(&stack);

printf("%d ", node->key);

if (node->right != NULL) push(&stack, node->right);

if (node->left != NULL) push(&stack, node->left);

}

}

void postorderTraversal(struct Node\* root) {

if (root == NULL) return;

struct Stack\* stack = NULL;

struct Stack\* outputStack = NULL;

push(&stack, root);

while (!isEmpty(stack)) {

struct Node\* node = pop(&stack);

push(&outputStack, node);

if (node->left != NULL) push(&stack, node->left);

if (node->right != NULL) push(&stack, node->right);

}

while (!isEmpty(outputStack)) {

struct Node\* node = pop(&outputStack);

printf("%d ", node->key);

}

}

int main() {

struct Node\* root = NULL;

int choice, key;

do {

printf("\nMenu:\n");

printf("1. Insert a node\n");

printf("2. Inorder traversal\n");

printf("3. Preorder traversal\n");

printf("4. Postorder traversal\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the node value to insert: ");

scanf("%d", &key);

if (root == NULL) {

root = newNode(key);

} else {

push(&root, key);

}

break;

case 2:

if (root == NULL) {

printf("The tree is empty.\n");

} else {

printf("Inorder traversal: ");

inorderTraversal(root);

printf("\n");

}

break;

case 3:

if (root == NULL) {

printf("The tree is empty.\n");

} else {

printf("Preorder traversal: ");

preorderTraversal(root);

printf("\n");

}

break;

case 4:

if (root == NULL) {

printf("The tree is empty.\n");

} else {

printf("Postorder traversal: ");

postorderTraversal(root);

printf("\n");

}

break;

case 5:

printf("Exiting...\n");

break;

default:

printf("Invalid choice, please try again.\n");

}

} while (choice != 5);

return 0;

}