**DR. D. Y. PATIL SCHOOL OF SCIENCE & TECHNOLOGY** 

**DR. D. Y. PATIL VIDYAPEETH, PUNE**

**(Deemed to be University)**

**(Accredited (3rd cycle) by NAAC with a CGPA of 3.64 on four-point scale at ‘A++’ Grade) (Declared as Category - I University by UGC Under Graded Autonomy Regulations, 2018) (An ISO 9001: 2015 and 14001:2015 Certified University and Green Education Campus)**

**Date:**

**Assignment No: 8**

**Problem Statement 1:**

**Design, Develop and Implement a menu driven Program in for the following operations on Binary Search Tree(BST) of Integers**

**Algorithm:**

1. Define the structure for a BST node with integer data and left and right pointers.
2. Define a function **createNode** to create a new node with the given integer value.
3. Define a function **insertNode** to insert a node into the BST recursively according to the BST property.
4. Define a function **inorderTraversal** to traverse the BST in inorder recursively and print the nodes.
5. Define a function **displayMenu** to display the menu options for the user.
6. In the **main** function:
   1. Declare a pointer to the root of the BST and initialize it to NULL.
   2. Display the menu in a do-while loop until the user chooses to exit.
   3. Inside the loop, prompt the user to enter a choice and perform the corresponding operation:
      1. If the choice is 1, prompt the user to enter a value and insert it into the BST using **insertNode.**
      2. If the choice is 2, perform an inorder traversal of the BST using **inorderTraversal**.
      3. If the choice is 3, exit the program.
      4. If the choice is invalid, display an error message.
7. Optionally, implement a function to free the memory allocated for BST nodes (**freeTree)** if necessary.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node \*createNode(int value) {

struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*insertNode(struct node \*root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insertNode(root->left, value);

} else if (value > root->data) {

root->right = insertNode(root->right, value);

}

return root;

}

void inorderTraversal(struct node \*root) {

if (root != NULL) {

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

}

void displayMenu() {

printf("\nBinary Search Tree Operations:\n");

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

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

printf("3. Exit\n");

}

int main() {

struct node \*root = NULL;

int choice, value;

do {

displayMenu();

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

root = insertNode(root, value);

break;

case 2:

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

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

break;

default:

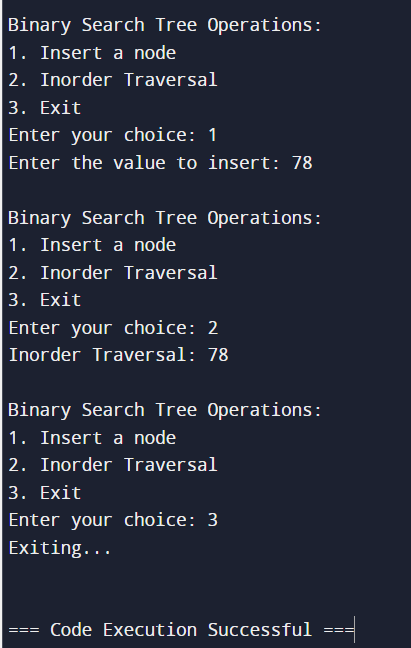
printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 3);

return 0;

}

**Sample Output:** 

**Problem Statement 2 :**

**Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2**

**Algorithm:**

1. Define the structure for a BST node with integer data and left and right pointers.
2. Define a function **createNode** to create a new node with the given integer value.
3. Define a function **insertNode** to insert a node into the BST recursively according to the BST property.
4. Create a BST with the given integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2.
   1. For each integer in the list:
      1. Insert the integer into the BST using the **insertNode** function.
5. Optionally, implement additional functions such as traversal or printing for verification.

**Sample Code :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node \*createNode(int value) {

struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*insertNode(struct node \*root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insertNode(root->left, value);

} else if (value > root->data) {

root->right = insertNode(root->right, value);

}

return root;

}

void inorderTraversal(struct node \*root) {

if (root != NULL) {

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

}

int main() {

struct node \*root = NULL;

root = insertNode(root, 6);

root = insertNode(root, 9);

root = insertNode(root, 5);

root = insertNode(root, 2);

root = insertNode(root, 8);

root = insertNode(root, 15);

root = insertNode(root, 24);

root = insertNode(root, 14);

root = insertNode(root, 7);

root = insertNode(root, 8);

root = insertNode(root, 5);

root = insertNode(root, 2);

printf("Inorder Traversal: ");

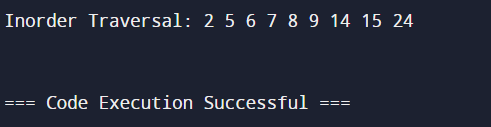
inorderTraversal(root);

printf("\n");

return 0;

}

**Sample Output :**



**Problem Statement 3 :**

**Traverse the BST in Inorder, Preorder and Post Order**

**Algorithm :**

**Inorder Traversal:**

* + Perform inorder traversal recursively.
  + For each node, traverse the left subtree.
  + Process the current node (print or store its value).
  + Traverse the right subtree.
* **Preorder Traversal:**
  + Perform preorder traversal recursively.
  + Process the current node (print or store its value).
  + Traverse the left subtree.
  + Traverse the right subtree.
* **Postorder Traversal:**
  + Perform postorder traversal recursively.
  + Traverse the left subtree.
  + Traverse the right subtree.
  + Process the current node (print or store its value).

**Sample Code :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node \*createNode(int value) {

struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*insertNode(struct node \*root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insertNode(root->left, value);

} else if (value > root->data) {

root->right = insertNode(root->right, value);

}

return root;

}

void inorderTraversal(struct node \*root) {

if (root != NULL) {

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

}

void preorderTraversal(struct node \*root) {

if (root != NULL) {

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

void postorderTraversal(struct node \*root) {

if (root != NULL) {

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ", root->data);

}

}

int main() {

struct node \*root = NULL;

root = insertNode(root, 6);

root = insertNode(root, 9);

root = insertNode(root, 5);

root = insertNode(root, 2);

root = insertNode(root, 8);

root = insertNode(root, 15);

root = insertNode(root, 24);

root = insertNode(root, 14);

root = insertNode(root, 7);

root = insertNode(root, 8);

root = insertNode(root, 5);

root = insertNode(root, 2);

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

printf("Postorder Traversal: ");

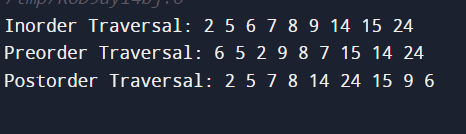
postorderTraversal(root);

printf("\n");

return 0;

}

**Sample Output :**

****

**Problem statement 4 : Search the BST for a given element (KEY) and report the appropriate message**

**Algorithm :**

1. **Start at the root of the BST.**
2. **Compare the key with the data of the current node.**
   1. **If the key is equal to the data of the current node, return "Found".**
   2. **If the key is less than the data of the current node, move to the left child.**
   3. **If the key is greater than the data of the current node, move to the right child.**

**3. Repeat steps 2 and 3 until you find the key or reach a leaf node.**

**4. If the key is not found after reaching a leaf node, return "Not Found".**

**Sample Code :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

struct node \*createNode(int value) {

struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*insertNode(struct node \*root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

root->left = insertNode(root->left, value);

} else if (value > root->data) {

root->right = insertNode(root->right, value);

}

return root;

}

const char \*searchKey(struct node \*root, int key) {

while (root != NULL) {

if (key == root->data) {

return "Found";

} else if (key < root->data) {

root = root->left;

} else {

root = root->right;

}

}

return "Not Found";

}

int main() {

struct node \*root = NULL;

root = insertNode(root, 6);

root = insertNode(root, 9);

root = insertNode(root, 5);

root = insertNode(root, 2);

root = insertNode(root, 8);

root = insertNode(root, 15);

root = insertNode(root, 24);

root = insertNode(root, 14);

root = insertNode(root, 7);

root = insertNode(root, 8);

root = insertNode(root, 5);

root = insertNode(root, 2);

int key = 14;

const char \*result = searchKey(root, key);

printf("Search for key %d: %s\n", key, result);

return 0;

}

**Sample Output :**

****