**Name :** Shubham Murlidhar Jadhav

**Roll No :** 26

**Class :** CS(AI-ML) -A

**Batch No** : 2

**PRN No :** 12210400

**Write a C program on AVL tree node insertion and Deletion operation.**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

int height;

}\*root;

*// Function to create a newNode*

struct Node\* createNode(int data)

{

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

newNode->height = 1;

return newNode;

}

*// Function to find max*

int max(int a, int b)

{

return a > b ? a:b;

}

*// Function to find height of a node*

int height(struct Node\* node)

{

if(node == NULL)

return 0;

return node->height;

}

*// Function to find Balance Factor*

int getBalance(struct Node\* N)

{

if(N == NULL)

return 0;

return height(N->left) - height(N->right);

}

*// Function to perform LeftRotation*

struct Node\* leftRotation(struct Node\* x)

{

struct Node\* y = x->right;

struct Node\* T2 = y->left;

*// Perform rotation*

y->left = x;

x->right = T2;

*// Update the heights*

x->height = 1 + max(height(x->left),height(x->right));

y->height = 1 + max(height(y->left),height(y->right));

return y;

}

*// Function to perform RightRotation*

struct Node\* rightRotation(struct Node\* y)

{

struct Node\* x = y->left;

struct Node\* T2 = x->right;

*// Perform rotation*

x->right = y;

y->left = T2;

*// Update the heights*

y->height = 1 + max(height(y->left),height(y->right));

x->height = 1 + max(height(x->left),height(x->right));

return x;

}

*// Function to create an AVL tree*

struct Node\* insertNode(struct Node\* root,int key)

{

if(root == NULL)

return createNode(key);

else if(key < root->data)

{

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

}

else if(key > root->data)

{

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

}

else return root;

*// Update the height of root*

root->height = 1 + max(height(root->left),height(root->right));

*// Check the balance factor of root is in the range or not*

int balance = getBalance(root);

*// Left Left case*

if(balance > 1 && key < root->left->data)

{

return rightRotation(root);

}

*// Right Right case*

else if(balance < -1 && key > root->right->data)

{

return leftRotation(root);

}

*// Left Right case*

else if(balance > 1 && key > root->left->data)

{

root->left = leftRotation(root->left);

return rightRotation(root);

}

*// Right Left case*

else if(balance < -1 && key < root->right->data)

{

root->right = rightRotation(root->right);

return leftRotation(root);

}

return root;

}

*// Function to find Inorder successor*

struct Node\* InSucc(struct Node\* node)

{

while(node && node->left != NULL)

{

node = node->left;

}

return node;

}

*// Function to find Inorder predecessor*

struct Node\* InPre(struct Node\* node)

{

while(node && node->right != NULL)

{

node = node->right;

}

return node;

}

struct Node\* DeleteNode(struct Node\* p,int key)

{

struct Node\* q = NULL;

if(p == NULL)

return NULL;

if(p->left == NULL && p->right == NULL)

{

if(p == root)

{

root = NULL;

}

free(p);

return NULL;

}

if(key < p->data)

{

p->left = DeleteNode(p->left,key);

}

else if(key > p->data)

{

p->right = DeleteNode(p->right,key);

}

*// Key found*

else

{

if(height(p->left) > height(p->right))

{

q = InPre(p->left);

p->data = q->data;

p->left = DeleteNode(p->left,q->data);

}

else

{

q = InSucc(p->right);

p->data = q->data;

p->right = DeleteNode(p->right,q->data);

}

}

*// Update the height of the current node*

p->height = 1 + max(height(p->left), height(p->right));

*// Get the balance factor of this node to check whether this node became unbalanced*

int balance = getBalance(p);

*// If this node becomes unbalanced, then there are 4 cases:*

*// Left Left Case*

if (balance > 1 && getBalance(p->left) >= 0)

return rightRotation(p);

*// Left Right Case*

if (balance > 1 && getBalance(p->left) < 0){

p->left = leftRotation(p->left);

return rightRotation(p);

}

*// Right Right Case*

if (balance < -1 && getBalance(p->right) <= 0)

return leftRotation(p);

*// Right Left Case*

if (balance < -1 && getBalance(p->right) > 0){

root->right = rightRotation(p->right);

return leftRotation(p);

}

return p;

}

*// Function to perfrom preOrder Traversal*

void preorderTraversal(struct Node\* root)

{

if(root == NULL)

return;

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

*// Function to perfrom inOrder Traversal*

void inorderTraversal(struct Node\* root)

{

if(root == NULL)

return;

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

*// Function to perfrom postOrder Traversal*

void postorderTraversal(struct Node\* root)

{

if(root == NULL)

return;

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

int main()

{

int choice, data, del;

root = NULL;

do {

printf("\nOptions:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display all Traversals\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nEnter data to insert: ");

scanf("%d", &data);

root = insertNode(root, data);

printf("Successfully Inserted!\n");

break;

case 2:

if (root == NULL) {

printf("Error: Tree is empty. Please insert some nodes first.\n");

break;

}

printf("\nEnter node to delete: ");

scanf("%d", &del);

root = DeleteNode(root, del);

printf("Successfully Deleted!\n");

break;

case 3:

if (root == NULL) {

printf("Error: Tree is empty. Please insert some nodes first.\n");

break;

}

printf("\nPreorder Traversal: ");

preorderTraversal(root);

printf("\n");

printf("\nInorder Traversal: ");

inorderTraversal(root);

printf("\n");

printf("\nPostorder Traversal: ");

postorderTraversal(root);

printf("\n");

break;

case 4:

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

break;

default:

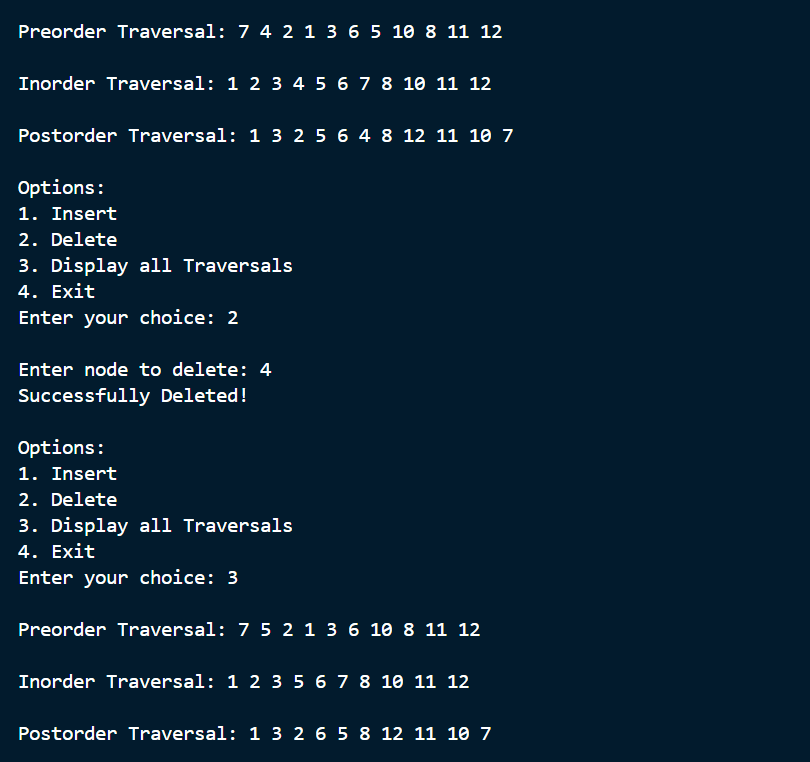
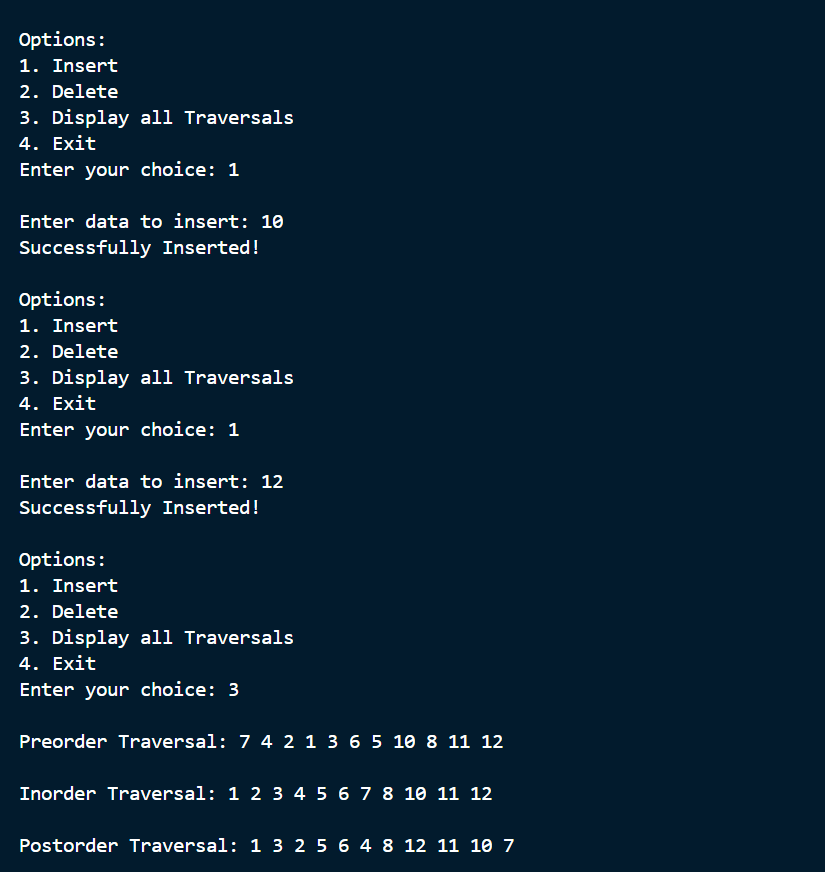
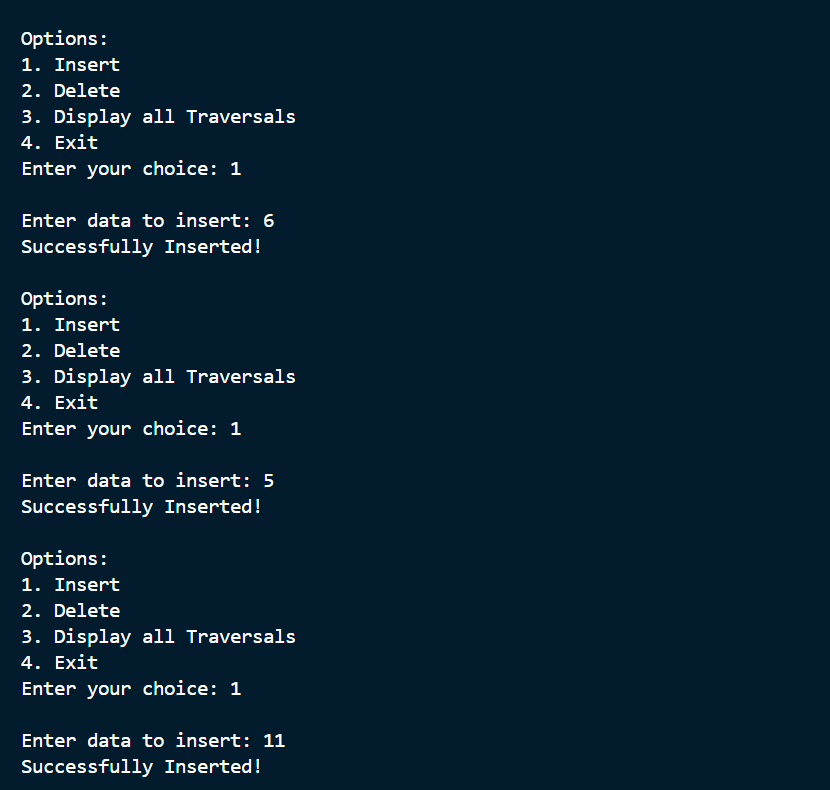
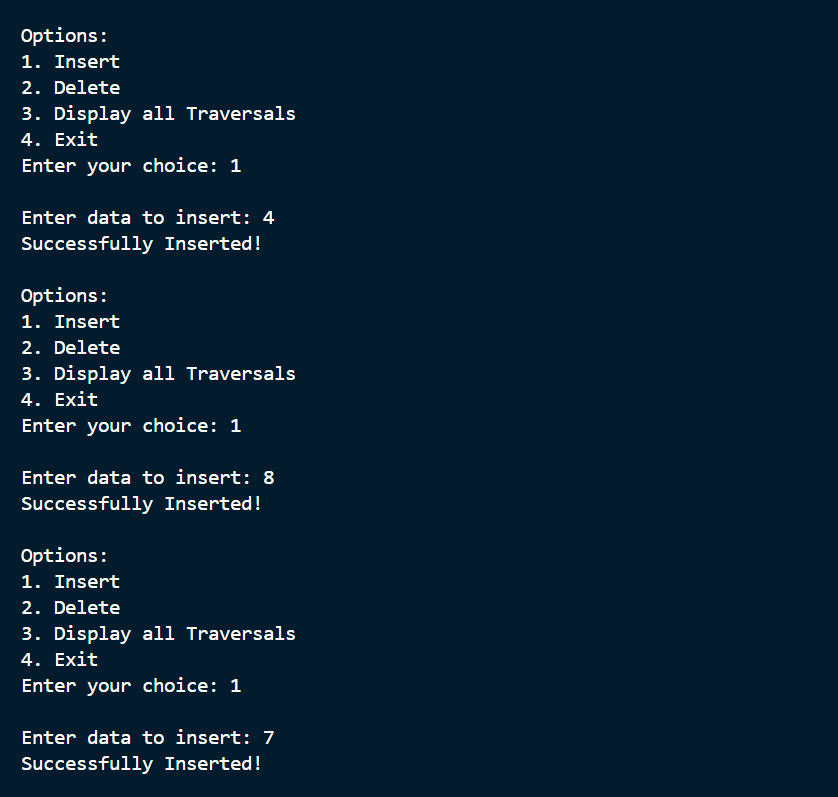
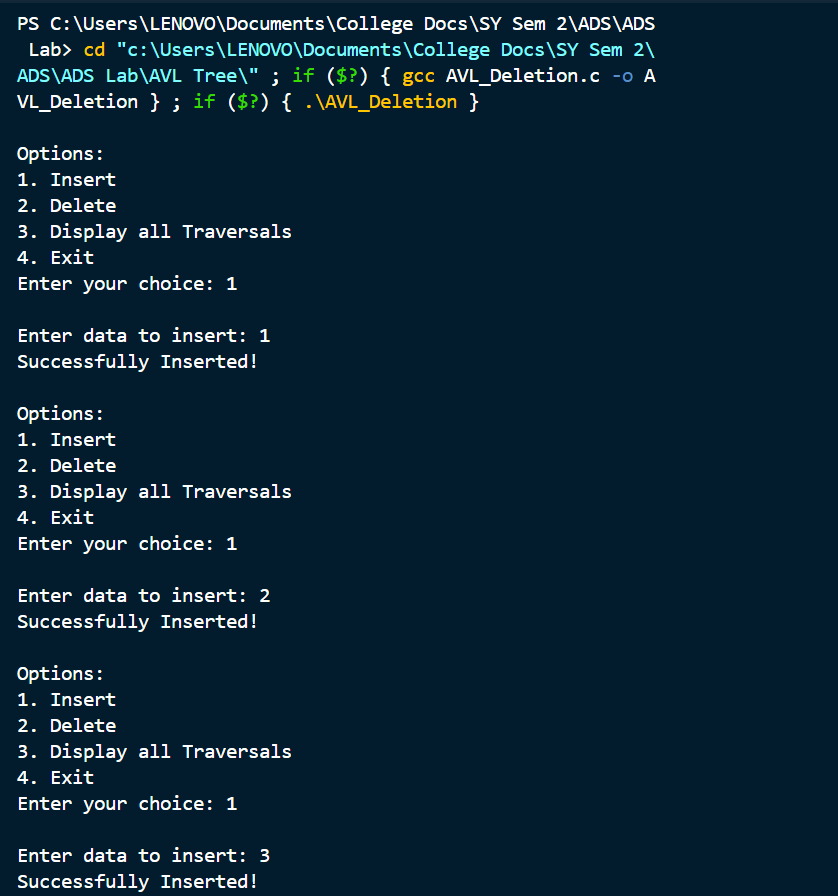
printf("Invalid choice. Please enter a valid option.\n");

}

} while(choice != 4);

return 0;

}

**Output :**