**Ds day9**

1.write a C program for binary tree:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* node, int data) {

if (node == NULL) {

return createNode(data);

}

if (data < node->data) {

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

} else {

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

}

return node;

}

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 = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("Inorder traversal of the binary tree: ");

inorderTraversal(root);

printf("\n");

return 0;

}

output:



2.write a C program for binary search tree:

// Binary Search Tree operations in C

#include <stdio.h>

#include <stdlib.h>

struct node {

int key;

struct node \*left, \*right;

};

// Create a node

struct node \*newNode(int item) {

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

temp->key = item;

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

return temp;

}

// Inorder Traversal

void inorder(struct node \*root) {

if (root != NULL) {

// Traverse left

inorder(root->left);

// Traverse root

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

// Traverse right

inorder(root->right);

}

}

// Insert a node

struct node \*insert(struct node \*node, int key) {

// Return a new node if the tree is empty

if (node == NULL) return newNode(key);

// Traverse to the right place and insert the node

if (key < node->key)

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

else

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

return node;

}

// Find the inorder successor

struct node \*minValueNode(struct node \*node) {

struct node \*current = node;

// Find the leftmost leaf

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

current = current->left;

return current;

}

// Deleting a node

struct node \*deleteNode(struct node \*root, int key) {

// Return if the tree is empty

if (root == NULL) return root;

// Find the node to be deleted

if (key < root->key)

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

else if (key > root->key)

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

else {

// If the node is with only one child or no child

if (root->left == NULL) {

struct node \*temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct node \*temp = root->left;

free(root);

return temp;

}

struct node \*temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

int main() {

struct node \*root = NULL;

root = insert(root, 8);

root = insert(root, 3);

root = insert(root, 1);

root = insert(root, 6);

root = insert(root, 7);

root = insert(root, 10);

root = insert(root, 14);

root = insert(root, 4);

printf("Inorder traversal: ");

inorder(root);

printf("\nAfter deleting 10\n");

root = deleteNode(root, 10);

printf("Inorder traversal: ");

inorder(root);

}

Output:

