//Program of preorder ,inorder and postorder of BST in C//

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

struct node \*root = NULL;

void insert(int data) {

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

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild = NULL;

if(root == NULL) {

root = tempNode;

}

else {

current = root;

parent = NULL;

while(1) {

parent = current;

if(data < parent->data) {

current = current->leftChild;

if(current == NULL) {

parent->leftChild = tempNode;

return;

}

}

else {

current = current->rightChild;

if(current == NULL) {

parent->rightChild = tempNode;

return;

}

}

}

}

}

struct node\* search(int data) {

struct node \*current = root;

printf("Visiting elements: ");

while(current->data != data) {

if(current != NULL)

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

if(current->data > data) {

current = current->leftChild;

}

else {

current = current->rightChild;

}

if(current == NULL) {

return NULL;

}

}

return current;

}

void pre\_order\_traversal(struct node\* root) {

if(root != NULL) {

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

pre\_order\_traversal(root->leftChild);

pre\_order\_traversal(root->rightChild);

}

}

void inorder\_traversal(struct node\* root) {

if(root != NULL) {

inorder\_traversal(root->leftChild);

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

inorder\_traversal(root->rightChild);

}

}

void post\_order\_traversal(struct node\* root) {

if(root != NULL) {

post\_order\_traversal(root->leftChild);

post\_order\_traversal(root->rightChild);

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

}

}

int main() {

int i;

int array[10] = { 13,27, 14, 35, 10, 19, 31, 42,33,42 };

for(i = 0; i < 7; i++)

insert(array[i]);

i = 31;

struct node \* temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

i = 15;

temp = search(i);

if(temp != NULL) {

printf("%d Element found.", temp->data);

printf("\n");

}else {

printf(" x Element not found (%d).\n", i);

} printf("Preorder traversal: ");

pre(root);

printf("Inorder traversal: ");

in(root);

printf("Post order traversal: ");

post(root);

return 0;

}

//C program to create and in order traversal on BST//

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

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

temp->key = item;

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

return temp;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

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

inorder(root->right);

}

}

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

{

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

if (key < node->key)

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

else if (key > node->key)

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

return node;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 4);

insert(root, 5);

insert(root, 9);

insert(root, 6);

insert(root, 3);

insert(root, 40);

insert(root, 41);

inorder(root);

return 0;

}

//Program in C of DFS//

#include<stdio.h>

void DFS(int);

int G[10][10],visited[10],n;

void main()

{

    int i,j;

    printf("Enter number of vertices:");

scanf("%d",&n);

    //read the adjecency matrix

printf("\nEnter adjecency matrix of the graph:");

for(i=0;i<n;i++)

       for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

    //visited is initialized to zero

   for(i=0;i<n;i++)

        visited[i]=0;

    DFS(0);

}

void DFS(int i)

{

    int j;

printf("\n%d",i);

    visited[i]=1;

for(j=0;j<n;j++)

       if(!visited[j]&&G[i][j]==1)

            DFS(j);

}

//Program in C of BFS//

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v) {

for (i=1;i<=n;i++)

if(a[v][i] && !visited[i])

q[++r]=i;

if(f<=r) {

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main() {

int v;

clrscr();

printf("\n Enter the number of vertices:");

scanf("%d",&n);

for (i=1;i<=n;i++) {

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

for (i=1;i<=n;i++)

for (j=1;j<=n;j++)

scanf("%d",&a[i][j]);

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

for (i=1;i<=n;i++)

if(visited[i])

printf("%d",i); else

printf("\n BFS is not possible");

getch();

}

//Program in C of linear search//

#include <stdio.h>

int main()

{

int a[50], a, i, n;

printf("Enter number of elements ");

scanf("%d", &n);

printf("Enter the integer\n");

for (i = 0; i < n; i++)

scanf("%d", &array[i]);

printf("Enter a number to search\n");

scanf("%d", &a);

for (i = 0; i < n; i++)

{

if (array[i] == a)

{

printf("%d is found at location %d.\n",a, i+1);

break;

}

}

if (i== n)

printf("%d isn't present in the array", search);

return 0;

}

//Program in C of binary search//

#include <stdio.h>

int main()

{

int i, f, la, mid, n, b, a[50];

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

for (i = 0; i < n; i++)

scanf("%d", &array[i]);

printf("Enter value to find\n");

scanf("%d", &b);

f = 0;

la= n - 1;

mid = (f+l)/2;

while (f <= la) {

if (a[mid] < b)

f = mid + 1;

else if (a[mid] == b)

{

printf("%d found at %d.\n", search, middle+1);

break;

}

else

la = mid - 1;

mid = (f+ l)/2;

}

if (f > l)

printf(Given number is not found in array");

return 0;

}