Data Structures LAB 10

Binary Trees Manual

**Name:** Ahmed Kasteer

**Roll Number**: 20F-0336

**Section**: 3D

**QUESTION 1:**

#include<iostream>

#define SPACE 10

using namespace std;

class TreeNode {

public:

int value;

TreeNode \* left;

TreeNode \* right;

TreeNode() {

value = 0;

left = NULL;

right = NULL;

}

};

class bst {

public:

TreeNode \* root;

bst() {

root = NULL;

}

bool isTreeEmpty() {

if (root == NULL) {

return true;

} else {

return false;

}

}

void insertNode(TreeNode \* new\_node) {

if (root == NULL) {

root = new\_node;

cout << "Value Inserted as root node!" << endl;

} else {

TreeNode \* temp = root;

while (temp != NULL) {

if (new\_node -> value == temp -> value) {

cout << "Value Already exist," <<

"Insert another value!" << endl;

return;

} else if ((new\_node -> value < temp -> value) && (temp -> left == NULL)) {

temp -> left = new\_node;

cout << "Value Inserted to the left!" << endl;

break;

} else if (new\_node -> value < temp -> value) {

temp = temp -> left;

} else if ((new\_node -> value > temp -> value) && (temp -> right == NULL)) {

temp -> right = new\_node;

cout << "Value Inserted to the right!" << endl;

break;

} else {

temp = temp -> right;

}

}

}

}

void printInorder(TreeNode \* r)

{

if (r == NULL)

return;

printInorder(r -> left);

cout << r -> value << " ";

printInorder(r -> right);

}

TreeNode \* minValueNode(TreeNode \* node) {

TreeNode \* current = node;

while (current -> left != NULL) {

current = current -> left;

}

return current;

}

TreeNode \* iterativeSearch(int v) {

if (root == NULL) {

return root;

} else {

TreeNode \* temp = root;

while (temp != NULL) {

if (v == temp -> value) {

return temp;

} else if (v < temp -> value) {

temp = temp -> left;

} else {

temp = temp -> right;

}

}

return NULL;

}

}

};

int main() {

cout<<"---QUESTION 1---\n";

bst b;

int option, val;

do {

cout << "What operation do you want to perform? " <<

" Select Option number. Enter 0 to exit." << endl;

cout << "1. Insert Node" << endl;

cout << "2. Search Node" << endl;

cout << "3. Print inorder" << endl;

cout << "0. Exit Program" << endl;

cin >> option;

TreeNode \* new\_node = new TreeNode();

switch (option) {

case 0:

break;

case 1:

cout <<"INSERT"<<endl;

cout <<"Enter VALUE of TREE NODE to INSERT in BST: ";

cin >> val;

new\_node->value = val;

b.insertNode(new\_node);

cout<<endl;

break;

case 2:

cout << "SEARCH" << endl;

cout << "Enter VALUE of TREE NODE to SEARCH in BST: ";

cin >> val;

new\_node = b.iterativeSearch(val);

if (new\_node != NULL) {

cout << "Value found" << endl;

} else {

cout << "Value NOT found" << endl;

}

break;

case 3:

cout << "PRINT inorder: " << endl;

cout <<"IN-ORDER: ";

b.printInorder(b.root);

cout<<endl;

break;

default:

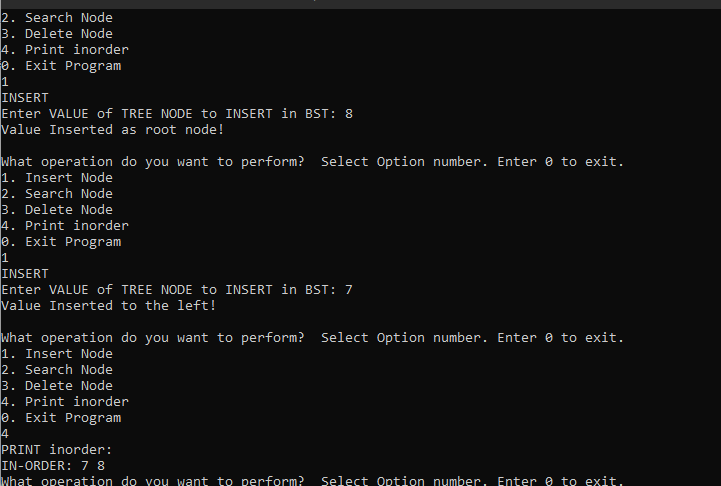
cout << "Enter Proper Option number " << endl;

}

} while (option != 0);

return 0;

}



**QUESTION 2:**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node\* left;

struct node\* right;

};

int \*merge(int array1[], int array2[], int m, int n);

void storeInorder(struct node\* node, int inorder[], int \*index\_ptr);

struct node\* sortedArrayToBST(int array[], int start, int end);

struct node\* mergeTrees(struct node \*root1, struct node \*root2, int m, int n)

{

int \*array1 = new int[m];

int i = 0;

storeInorder(root1, array1, &i);

int \*array2 = new int[n];

int j = 0;

storeInorder(root2, array2, &j);

int \*mergedArr = merge(array1, array2, m, n);

return sortedArrayToBST (mergedArr, 0, m+n-1);

}

struct node\* newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return(node);

}

void printInorder(struct node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

cout<< node->data;

printInorder(node->right);

}

int \*merge(int array1[], int array2[], int m, int n)

{

int \*mergedArr = new int[m + n];

int i = 0, j = 0, k = 0;

while (i < m && j < n)

{

if (array1[i] < array2[j])

{

mergedArray[k] = array1[i];

i++;

}

else

{

mergedArr[k] = arr2[j];

j++;

}

k++;

}

while (i < m)

{

mergedArray[k] = array1[i];

i++; k++;

}

while (j < n)

{

mergedArray[k] = array2[j];

j++; k++;

}

return mergedArray;

}

void storeInorder(struct node\* node, int inorder[], int \*index\_ptr)

{

if (node == NULL)

return;

storeInorder(node->left, inorder, index\_ptr);

inorder[\*index\_ptr] = node->data;

(\*index\_ptr)++;

storeInorder(node->right, inorder, index\_ptr);

}

struct node\* sortedArrayToBST(int arr[], int start, int end)

{

if (start > end)

return NULL;

int mid = (start + end)/2;

struct node \*root = newNode(arr[mid]);

root->left = sortedArrayToBST(array, start, mid-1);

root->right = sortedArrayToBST(array, mid+1, end);

return root;

}

int main()

{

struct node \*root1 = newNode(150);

root1->left = newNode(60);

root1->right = newNode(250);

root1->left->left = newNode(20);

root1->left->right = newNode(70);

struct node \*root2 = newNode(90);

root2->left = newNode(45);

root2->right = newNode(130);

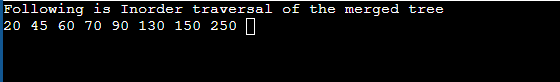
struct node \*mergedTree = mergeTrees(root1, root2, 5, 3);

printf ("Following is Inorder traversal of the merged tree \n");

printInorder(mergedTree);

return 0;

}



**QUESTION 3:**

#include<iostream>

#define SPACE 10

using namespace std;

class TreeNode {

public:

int value;

TreeNode \* left;

TreeNode \* right;

TreeNode() {

value = 0;

left = NULL;

right = NULL;

}

};

class bst {

public:

TreeNode \* root;

bst() {

root = NULL;

}

bool isTreeEmpty() {

if (root == NULL) {

return true;

} else {

return false;

}

}

void insertNode(TreeNode \* new\_node) {

if (root == NULL) {

root = new\_node;

cout << "Value Inserted as root node!" << endl;

} else {

TreeNode \* temp = root;

while (temp != NULL) {

if (new\_node -> value == temp -> value) {

cout << "Value Already exist," <<

"Insert another value!" << endl;

return;

} else if ((new\_node -> value < temp -> value) && (temp -> left == NULL)) {

temp -> left = new\_node;

cout << "Value Inserted to the left!" << endl;

break;

} else if (new\_node -> value < temp -> value) {

temp = temp -> left;

} else if ((new\_node -> value > temp -> value) && (temp -> right == NULL)) {

temp -> right = new\_node;

cout << "Value Inserted to the right!" << endl;

break;

} else {

temp = temp -> right;

}

}

}

}

void printInorder(TreeNode \* r)

{

if (r == NULL)

return;

printInorder(r -> left);

cout << r -> value << " ";

printInorder(r -> right);

}

TreeNode \* minValueNode(TreeNode \* node) {

TreeNode \* current = node;

while (current -> left != NULL) {

current = current -> left;

}

return current;

}

TreeNode \* iterativeSearch(int v) {

if (root == NULL) {

return root;

} else {

TreeNode \* temp = root;

while (temp != NULL) {

if (v == temp -> value) {

return temp;

} else if (v < temp -> value) {

temp = temp -> left;

} else {

temp = temp -> right;

}

}

return NULL;

}

}

TreeNode \* deleteNode(TreeNode \* r, int v) {

if (r == NULL) {

return NULL;

}

else if (v < r -> value) {

r -> left = deleteNode(r -> left, v);

}

else if (v > r -> value) {

r -> right = deleteNode(r -> right, v);

}

else {

if (r -> left == NULL) {

TreeNode \* temp = r -> right;

delete r;

return temp;

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

TreeNode \* temp = r -> left;

delete r;

return temp;

} else {

TreeNode \* temp = minValueNode(r -> right);

r -> value = temp -> value;

r -> right = deleteNode(r -> right, temp -> value);

}

}

return r;

}

};

int main() {

cout<<"---QUESTION 3---\n";

bst b;

int option, val;

do {

cout << "What operation do you want to perform? " <<

" Select Option number. Enter 0 to exit." << endl;

cout << "1. Insert Node" << endl;

cout << "2. Search Node" << endl;

cout << "3. Delete Node" << endl;

cout << "4. Print inorder" << endl;

cout << "0. Exit Program" << endl;

cin >> option;

TreeNode \* new\_node = new TreeNode();

switch (option) {

case 0:

break;

case 1:

cout <<"INSERT"<<endl;

cout <<"Enter VALUE of TREE NODE to INSERT in BST: ";

cin >> val;

new\_node->value = val;

b.insertNode(new\_node);

cout<<endl;

break;

case 2:

cout << "SEARCH" << endl;

cout << "Enter VALUE of TREE NODE to SEARCH in BST: ";

cin >> val;

new\_node = b.iterativeSearch(val);

if (new\_node != NULL) {

cout << "Value found" << endl;

} else {

cout << "Value NOT found" << endl;

}

break;

case 3:

cout << "DELETE" << endl;

cout << "Enter VALUE of TREE NODE to DELETE in BST: ";

cin >> val;

new\_node = b.iterativeSearch(val);

if (new\_node != NULL) {

b.deleteNode(b.root, val);

cout << "Value Deleted" << endl;

} else {

cout << "Value NOT found" << endl;

}

break;

case 4:

cout << "PRINT inorder: " << endl;

cout <<"IN-ORDER: ";

b.printInorder(b.root);

cout<<endl;

break;

default:

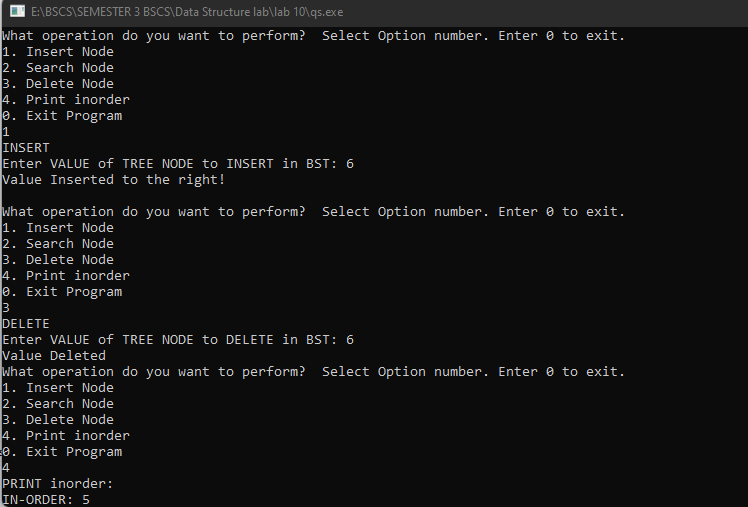
cout << "Enter Proper Option number " << endl;

}

} while (option != 0);

return 0;

}



**QUESTION 7:**

**QUESTION 5:**

#include <iostream>

using namespace std;

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

node\* temp=new node;

temp->key = item;

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

return temp;

}

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

{

struct node \*newNode(int );

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 minValue(struct node\* node)

{

struct node\* current = node;

while (current->left != NULL)

{

current = current->left;

}

return(current->key);

}

int maxValue(struct node\* node)

{

struct node\* current = node;

while (current->right != NULL)

{

current = current->right;

}

return(current->key);

}

int main()

{

cout<<"----QUESTION 5----\n";

int maxValue(struct node\* );

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

int minValue(struct node\* );

struct node \*root = NULL;

root = insert(root, 11);

insert(root, 9);

insert(root, 22);

insert(root, 3);

insert(root, 4);

insert(root, 7);

insert(root, 8);

insert(root, 10);

insert(root, 9);

insert(root, 13);

insert(root, 14);

insert(root, 19);

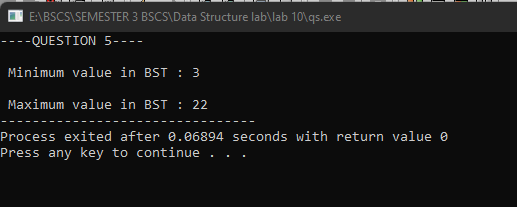
insert(root, 5);

cout << "\n Minimum value in BST : " << minValue(root)<<endl;

cout << "\n Maximum value in BST : " << maxValue(root);

return 0;

}



**QUESTION 6:**

**QUESTION 7:**

#include <iostream>

#include <stack>

using namespace std;

struct Node

{

int data;

Node \*left, \*right;

Node(int data)

{

this->data = data;

this->left = this->right = nullptr;

}

};

void preorder\_Iterative(Node\* root)

{

if (root == nullptr)

return;

stack<Node\*> stack;

stack.push(root);

while (!stack.empty())

{

Node\* curr = stack.top();

stack.pop();

cout << curr->data << " ";

if (curr->right) {

stack.push(curr->right);

}

if (curr->left) {

stack.push(curr->left);

}

}

}

void postorder\_Iterative(Node\* root)

{

cout<<endl;

if (root == nullptr) {

return;

}

stack<Node\*> s;

s.push(root);

stack<int> out;

while (!s.empty())

{

Node\* curr = s.top();

s.pop();

out.push(curr->data);

if (curr->left) {

s.push(curr->left);

}

if (curr->right) {

s.push(curr->right);

}

}

while (!out.empty())

{

cout << out.top() << " ";

out.pop();

}

}

int main()

{

cout<<"---QUESTION 7---\n";

Node\* root = new Node(9);

root->left = new Node(8);

root->right = new Node(7);

root->left->left = new Node(6);

root->right->left = new Node(5);

root->right->right = new Node(4);

root->right->left->left = new Node(3);

root->right->left->right = new Node(2);

cout<<"Preorder iterative traversal :\n";

preorder\_Iterative(root);

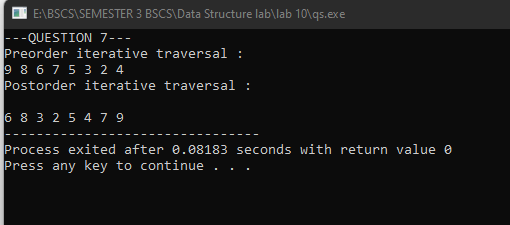
cout<<endl;

cout<<"Postorder iterative traversal :\n";

postorder\_Iterative(root);

return 0;

}



**QUESTION 8:**

#include<iostream>

#include<limits.h>

using namespace std;

struct Node

{

int data;

Node\* left;

Node\* right;

Node(int val)

{

data = val;

left = NULL;

right = NULL;

}

};

int isBST\_til(Node\* node, int min, int max)

{

if (node==NULL)

return 1;

if (node->data < min || node->data > max)

return 0;

return

isBSTUtil(node->left, min, node->data-1) &&

isBSTUtil(node->right, node->data+1, max);

}

int isBstTree(Node\* rt)

{

return(isBSTUtil(rt, INT\_MIN, INT\_MAX));

}

int main()

{

cout<<"---QUESTION 8---\n";

Node\* root = new Node(11);

root->left = new Node(7);

root->right = new Node(14);

root->left->left = new Node(2);

root->left->right = new Node(9);

if(isBstTree(root))

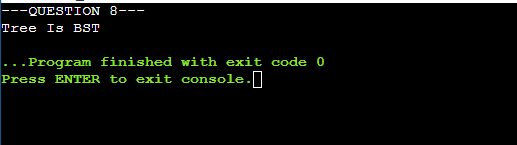
cout<<"Tree Is BST";

else

cout<<"Tree is Not a BST";

return 0;

}



**QUESTION 9:**

#include <iostream>

using namespace std;

struct node

{

int data;

struct node\* left;

struct node\* right;

};

int countLeafNode(struct node\* node)

{

if(node == NULL)

return 0;

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

return 1;

else

return countLeafNode(node->left)+countLeafNode(node->right);

}

struct node\* newNode(int data)

{

struct node\* node = (struct node\*)

malloc(sizeof(struct node));

node->data = data;

node->left = NULL;

node->right = NULL;

return(node);

}

int main()

{

cout<<"--QUESTION 9---\n";

struct node \*root = newNode(2);

root->left = newNode(8);

root->right = newNode(9);

root->left->left = newNode(3);

root->left->right = newNode(4);

cout << "Leaf count of the tree is : "<<

countLeafNode(root) << endl;

return 0;

}

