**TASK 1://using Iterations**

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left;

node\* right;

};

node\* root = NULL;

class BST

{

int data;

node\* left;

node\* right;

public:

BST()

{

}

void Insert(int data)

{

int counter = 0;

node\* newNode = new node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

if (root == NULL)

{

root = newNode;

return;

}

node\* N = root;

counter++;

// i am placing one on left and next one on right

if (counter % 2 == 0)

{

InsertLeft(data);

}

else

{

InsertRight(data);

}

}

void InsertLeft(int data)

{

node\* newNode = new node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

if (root == NULL)

{

root = newNode;

return;

}

if (root->left == NULL)

{

root->left = newNode;

return;

}

node\* N = root;

N = N->left;

while (N != NULL)

{

if (N->left == NULL)

{

N->left = newNode;

return;

}

else

{

N->right = newNode;

return;

}

N = N->left;

}

}

void InsertRight(int data)

{

node\* newNode = new node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

if (root == NULL)

{

root = newNode;

return;

}

if (root->right == NULL)

{

root->right = newNode;

return;

}

node\* N = root;

N = N->right;

while (N != NULL)

{

if (N->right == NULL)

{

N->right = newNode;

return;

}

else

{

N->left = newNode;

return;

}

N = N->right;

}

}

bool search(int data)

{

node\* newnode = new node;

newnode->data = data;

newnode->left = NULL;

newnode->right = NULL;

newnode = root;

while (newnode != NULL)

{

if (data > root->data)

{

newnode = newnode->right;

}

if (data < root->data)

{

newnode = newnode->left;

}

if(data==root->data)

{

return true;

break;

}

}

return false;

}

bool is\_empry()

{

if (root == NULL)

{

return true;

}

else

{

return false;

}

}

node\* delete\_Data(node\* root, int data)

{

node\* newnode = new node;

newnode->data = data;

newnode->left = NULL;

newnode->right = NULL;

newnode = root;

while (newnode != NULL)

{

if (data > root->data)

{

newnode = newnode->right;

}

if (data < root->data)

{

newnode = newnode->left;

}

if(data==root->data)

{

return true;

delete (root->data);

break;

}

}

return false;

}

}

};

int main()

{

BST bst;

int num, data, key, n;

node\* root = NULL;

int op;

a: cout << "Enter 1 for insertion :"<<endl;

cout << "Enter 2 for deletion :" << endl;

cout << "Enter 3 for searching :" << endl;

cout << "Enter 4 for checking if it is empty :" << endl;

cout << "Enter 5 to exit :" << endl;

cin >> op;

if (op == 1)

{

cout << "Enter how many node you want in a BST :" << endl;

cin >> num;

for (int i = 0; i < num; i++)

{

cin >> data;

bst.Insert(data);

}

goto a;

}

else if (op == 2)

{

cout << "Enter number you want to delete in the tree :";

cin >> n;

bst.delete\_Data(root, n);

goto a;

}

else if (op == 3)

{

cout << "Enter number you want to search in the tree :";

cin >> key;

if (!bst.search(key))

{

cout << "Key has been found" << endl;

}

else

{

cout << "Key has found" << endl;

}

goto a;

}

else if (op == 4)

{

if (bst.is\_empry())

{

cout << "Tree is empty "<<endl;

}

else

{

cout << "Tree is not empty "<<endl;

}

goto a;

}

else if(op==5)

{

exit;

}

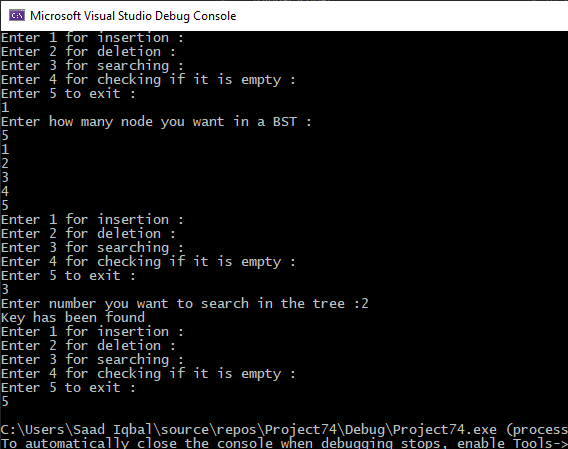
else

{

goto a;

}

}



**TASK 2://using RECURSION**

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left, \* right;

};

node\* root = NULL;

class Tree

{

private:

node\* Root = NULL;

int counter = 0;

public:

node\* getRoot()

{

return Root;

}

void Display(node\* N, int space)

{

if (N == NULL)

return;

space += 10;

Display(N->right, space);

cout << endl;

for (int i = 10; i < space; i++)

cout << " ";

cout << N->data << "\n";

Display(N->left, space);

}

node\* insert(int x, node\* t)

{

if (t == NULL)

{

t = new node;

t->data = x;

t->left = t->right = NULL;

}

else if (x < t->data)

{

t->left = insert(x, t->left);

}

else if (x > t->data)

{

t->right = insert(x, t->right);

}

return t;

}

bool Search(node\* N, int data)

{

if (N == NULL)

{

return N;

}

if (data == N->data)

{

return true;

}

return Search(N->right, data);

return Search(N->left, data);

}

node\* delete\_data(node\* root, int data)

{

if (root == NULL)

{

return NULL;

}

else if (root->data == data)

{

free(root->left);

free(root->right);

free(root);

root = NULL;

}

else if (data < root->data)

{

root->left = delete\_data(root->left, data);

}

else if (data > root->data)

{

root->right = delete\_data(root->right, data);

}

else

{

if (root->left == NULL)

{

node\* temp = root->right;

delete root;

return temp;

}

if (root->right == NULL)

{

node\* temp = root->left;

delete root;

return temp;

}

else

{

node\* temp = root->left;

root->data = temp->data;

root->right = delete\_data(root->right, temp->data);

}

}

}

bool isempty()

{

node\* t = new node;

t = root;

if (root == NULL)

{

return true;

}

else

{

return false;

}

}

};

int main()

{

Tree bst;

Tree \*bst1=NULL;

int num, data, key, n;

node\* root = NULL;

int op;

a: cout << "Enter 1 for insertion :" << endl;

cout << "Enter 2 for deletion :" << endl;

cout << "Enter 3 for searching :" << endl;

cout << "Enter 4 for checking if it is empty :" << endl;

cout << "Enter 5 to exit :" << endl;

cin >> op;

if (op == 1)

{

cout << "Enter how many node you want in a BST :" << endl;

cin >> num;

for (int i = 0; i < num; i++)

{

cin >> data;

bst.insert(data,root);

}

goto a;

}

else if (op == 2)

{

cout << "Enter number you want to delete in the tree :";

cin >> n;

bst1->delete\_data(root, n);

goto a;

}

else if (op == 3)

{

cout << "Enter number you want to search in the tree :";

cin >> key;

if (!bst.Search(root,key))

{

cout << "Key has been found" << endl;

}

else

{

cout << "Key has found" << endl;

}

goto a;

}

else if (op == 4)

{

if (bst.isempty())

{

cout << "Tree is empty " << endl;

}

else

{

cout << "Tree is not empty " << endl;

}

goto a;

}

else if (op == 5)

{

exit;

}

else

{

goto a;

}

system("pause");

}

