Following is the Binary Tree Node class structure

template <typename T>

class BinaryTreeNode {

public:

T data;

BinaryTreeNode<T> \*left;

BinaryTreeNode<T> \*right;

BinaryTreeNode(T data) {

this->data = data;

left = NULL;

right = NULL;

}

};

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class BST {

// Define the data members

BinaryTreeNode<int> \*root;

public:

BST() {

// Implement the Constructor

root = NULL;

}

~BST()

{

delete root;

}

private:

BinaryTreeNode<int>\* helperinsert(int data , BinaryTreeNode<int> \*root)

{

if(root == NULL)

{

BinaryTreeNode<int> \*newnode = new BinaryTreeNode<int>(data);

return newnode;

}

if(data > root->data)

{

BinaryTreeNode<int> \*ansr = helperinsert(data , root->right);

root->right = ansr;

}

if (data <= root->data)

{

BinaryTreeNode<int> \*ansl = helperinsert(data , root->left);

root->left = ansl;

}

return root;

}

// bool helpersearch(int data , BinaryTreeNode<int> \*root)

// {

// if(root == NULL)

// {

// return false;

// }

// if(root->data == data)

// {

// return true;

// }

// if(data > root->right->data)

// {

// return helpersearch(data , root->right);

// }

// else if(data < root->left->data)

// {

// return helpersearch(data , root->right);

// }

// }

bool helpersearch(int data,BinaryTreeNode<int> \*root) // helper function for searching

{

if(root==NULL)

return false;

if(root->data == data)

{

return true;

}

if(root->data > data)

{

return helpersearch(data,root->left);

}

if(root->data < data)

{

return helpersearch(data,root->right);

}

}

void helperprint(BinaryTreeNode<int> \*root)

{

if(root == NULL)

{

return;

}

cout<<root->data<<":";

if(root->left != NULL)

{

cout<<"L:"<<root->left->data<<",";

}

if(root->right != NULL)

{

cout<<"R:"<<root->right->data;

}

cout<<endl;

helperprint(root->left);

helperprint(root->right);

}

BinaryTreeNode<int>\* smallRight(BinaryTreeNode<int> \*root)

{

if(root == NULL)

{

return NULL;

}

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

{

return root;

}

BinaryTreeNode<int> \*ans = smallRight(root->left);

return ans;

}

BinaryTreeNode<int>\* helperdelete(int data , BinaryTreeNode<int> \*root)

{

if(root == NULL)

{

return NULL;

}

if(root->data < data){

BinaryTreeNode<int> \*r = helperdelete(data , root->right);

root->right = r;

return root;// here

}

if(root->data > data)

{

BinaryTreeNode<int> \*l = helperdelete(data , root->left);

root->left = l;

return root;// here

}

if(root->data == data)

{

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

{

delete root;

return NULL;

}

if(root->left != NULL && root->right == NULL)

{

BinaryTreeNode<int> \*temp = root->left;

root->left = NULL;

delete root;

return temp;

}

else if(root->left == NULL && root->right != NULL)

{

BinaryTreeNode<int> \*temp = root->right;

root->right = NULL;

delete root;

return temp;

}

else if(root->left != NULL && root->right != NULL)

{

BinaryTreeNode<int> \*smallest = smallRight(root->right);

int rightmin = smallest->data;

root->data = rightmin;

root->right = helperdelete(smallest->data , root->right);

return root;

}

}

}

/\*----------------- Public Functions of BST -----------------\*/

public:

void remove(int data) {

// Implement the remove() function

BinaryTreeNode<int> \*ans = helperdelete(data , root);

root = ans;

}

void print() {

// Implement the print() function

helperprint(root);

}

void insert(int data) {

// Implement the insert() function

BinaryTreeNode<int> \*insertnode = helperinsert(data , root);

root = insertnode;

}

bool search(int data) {

// Implement the search() function

return helpersearch(data , root);

}

};

int main() {

BST \*tree = new BST();

int choice, input, q;

cin >> q;

while (q--) {

cin >> choice;

switch (choice) {

case 1:

cin >> input;

tree->insert(input);

break;

case 2:

cin >> input;

tree->remove(input);

break;

case 3:

cin >> input;

cout << ((tree->search(input)) ? "true\n" : "false\n");

break;

default:

tree->print();

break;

}

}

}