#include <iostream>

using namespace std;

class Node

{

public:

string name;

string meaning;

Node\* left;

Node\* right;

Node\* temp;

Node()

{

left = NULL;

right = NULL;

}

Node(string name, string meaning)

{

this->name = name;

this->meaning = meaning;

left = NULL;

right = NULL;

}

};

class Dictonary

{

public:

Node\* root;

Dictonary()

{

root = NULL;

}

void createfunc();

Node\* insertfunc(Node\*, string, string);

void Inorder(Node\*);

void Postorder(Node\*);

void Preorder(Node\*);

void Search(Node\*, string);

Node\* DeleteNode(Node\*, string);

};

void Dictonary::createfunc()

{

int t;

cout << "Enter Number of Nodes" << endl;

cin >> t;

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

{

string iword, imeaning;

cout << "Enter word" << endl;

cin >> iword;

cout << "Enter the Meaning here" << endl;

cin >> imeaning;

root = insertfunc(root, iword, imeaning);

}

cout << "1: For Inorder" << endl;

cout << "2: For Postorder" << endl;

cout << "3: For Preorder" << endl;

cout << "4: For Searching" << endl;

int yb;

cin >> yb;

if (yb == 1)

{

Inorder(root);

}

else if (yb == 2)

{

Postorder(root);

}

else if (yb == 3)

{

Preorder(root);

}

else if (yb == 4)

{

cout << "Enter data to be searched" << endl;

string temp;

cin >> temp;

Search(root, temp);

}

}

Node\* Dictonary::insertfunc(Node\* temproot, string w, string m) {

Node\* temp = new Node(w, m);

if (temproot == NULL)

{

return temp;

}

if (w < temproot->name)

{

cout << "IN" << endl;

temproot->left = insertfunc(temproot->left, w, m);

}

else

{

temproot->right = insertfunc(temproot->right, w, m);

}

return temproot;

}

void Dictonary::Inorder(Node\* root)

{

if (root == NULL)

{

return;

}

Inorder(root->left);

cout << root->name << endl;

Inorder(root->right);

}

void Dictonary::Postorder(Node\* root)

{

if (root == NULL)

{

return;

}

Postorder(root->left);

Postorder(root->right);

cout << root->name << ":" << root->meaning << endl;

}

void Dictonary::Preorder(Node\* root)

{

if (root == NULL)

{

return;

}

cout << root->name << ":" << root->meaning << endl;

Preorder(root->left);

Preorder(root->right);

}

void Dictonary::Search(Node\* root, string temp)

{

if (root == NULL)

{

cout << "Not Found" << endl;

return;

}

if (root->name == temp)

{

cout << temp << "Found" << endl;

}

else if (root->name > temp)

{

Search(root->left, temp);

}

else

{

Search(root->right, temp);

}

}

Node\* Dictonary::DeleteNode(Node\* root, string temp)

{

if (root == NULL)

{

cout << "Not found";

return NULL;

}

else if (root->name == temp)

{

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

{

delete root;

return NULL;

}

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

{

return root->left;

}

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

{

return root->right;

}

}

else if (root->name > temp)

{

DeleteNode(root->left, temp);

}

else

{

DeleteNode(root->right, temp);

}

}

int main()

{

Dictonary d;

d.createfunc();

return 0;

}