#include<iostream>

using namespace std;

struct Node

{

string key;

string meaning;

Node\* left;

Node\* right;

};

class AVL{

Node\* root;

public:

AVL(){

root=NULL;

}

void create();

Node\*insert(Node\* curr,Node\* temp);

Node\* balance(Node\* temp);

int diff(Node\* temp);

int height(Node\* root);

int maximum(int a,int b);

Node\* ll(Node\* parent);

Node\* rr(Node\* parent);

Node\* lr(Node\* parent);

Node\* rl(Node\* parent);

void ascending(Node\* temp);

void descending(Node\* temp);

void display();

bool search(Node\* curr,string key1);

void search\_value();

};

void AVL::create(){

Node\* temp;

char answer;

do{

temp=new Node;

cout<<"Enter keyword"<<"\n";

cin>>temp->key;

cout<<"Enter meaning"<<"\n";

cin>>temp->meaning;

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

root=insert(root,temp);

cout<<"Do you want to add more nodes"<<"\n";

cin>>answer;

//root=insert(root,temp);

}while(answer=='Y'||answer=='y');

}

Node\* AVL::insert(Node\* curr,Node\* temp){

if(curr==NULL){

return temp;

}

if(curr->key>temp->key){

curr->left=insert(curr->left,temp);

curr=balance(curr);

}

else if(curr->key<temp->key){

curr->right=insert(curr->right,temp);

curr=balance(curr);

}

return curr;

}

Node\* AVL::balance(Node\* temp){

int bal;

bal=diff(temp);

if(bal>=2){

if(diff(temp->left)<0){

temp=lr(temp);

}

else{

temp=ll(temp);

}

}

else if((bal<=-2)){

if(diff(temp->right)<0){

temp=rr(temp);

}

else{

temp=rl(temp);

}

}

return temp;

}

int AVL::diff(Node\* temp){

int l=height(temp->left);

int r=height(temp->right);

return(l-r);

}

int AVL::height(Node\* root){

if(root==NULL){

return(-1);

}

else{

return(1+max(height(root->left),height(root->right)));

}

}

int AVL::maximum(int a,int b){

if(a>b){

return a;

}

else{

return b;

}

}

Node\* AVL::ll(Node\* parent){

Node\* temp;

Node\* temp1;

temp=parent->left;

temp1=temp->right;

temp->right=parent;

parent->left=temp1;

return temp;

}

Node\* AVL::rr(Node\* parent){

Node\* temp;

Node\* temp1;

temp=parent->right;

temp1=temp->left;

temp->left=parent;

parent->right=temp1;

return temp;

}

Node\* AVL::lr(Node\* parent){

parent->left=rr(parent->left);

return(ll(parent));

}

Node\* AVL::rl(Node\* parent){

parent->right=ll(parent->right);

return(rr(parent));

}

void AVL::ascending(Node\* temp){

if(temp!=NULL){

ascending(temp->left);

cout<<"keyword"<<" "<<temp->key<<" "<<"meaning"<<" "<<temp->meaning<<"\n";

ascending(temp->right);

}

}

void AVL::descending(Node\* temp){

if(temp!=NULL){

descending(temp->right);

cout<<"keyword"<<" "<<temp->key<<" "<<"meaning"<<" "<<temp->meaning<<"\n";

descending(temp->left);

}

}

void AVL::display(){

cout<<"Keywords in ascending order"<<"\n";

ascending(root);

cout<<"Keywords in descending order"<<"\n";

descending(root);

}

bool AVL::search(Node\* curr,string key1){

if(curr->key==key1)

return true;

else if(curr->key>key1){

return search(curr->left,key1);

}

else{

return search(curr->right,key1);

}

}

void AVL::search\_value(){

string key1;

cout<<"enter keyword to search"<<"\n";

cin>>key1;

if(search(root,key1)){

cout<<"Keyword found"<<"\n";

}

else{

cout<<"Keyword not found"<<"\n";

}

}

int main(){

int choice,ch;

Node\* temp;

Node\* root;

AVL a;

do{

cout<<"1.create tree"<<"\n";

//cout<<"2.insert tree"<<"\n";

cout<<"2.display"<<"\n";

cout<<"3.search"<<"\n";

cin>>choice;

switch(choice){

case 1:

a.create();

break;

case 2:

a.display();

break;

case 3:

a.search\_value();

}

cout<<"Do you want continue"<<"\n";

cin>>ch;

}while(ch==4);

}