#include<iostream>

#include<string>

using namespace std;

class dictionary;

class avlnode

{

string keyword;

string meaning;

avlnode \*left,\*right;

int bf;

public:

avlnode()

{

keyword='\0';

meaning='\0';

left=right=NULL;

bf=0;

}

avlnode(string k,string m)

{

keyword=k;

meaning=m;

left=right=NULL;

bf=0;

}

friend class dictionary;

};

class dictionary

{

avlnode \*par,\*loc;

public:

avlnode \*root;

dictionary()

{

root=NULL;

par=loc=NULL;

}

void accept();

void insert(string key,string mean);

void LLrotation(avlnode\*,avlnode\*);

void RRrotation(avlnode\*,avlnode\*);

void inorder(avlnode \*root);

void deletekey(string key);

void descending(avlnode \*);

void search(string);

void update(string,string);

};

void dictionary::descending(avlnode \*root)

{

if(root)

{

descending(root->right);

cout<<root->keyword<<" "<<root->meaning<<endl;

descending(root->left);

}

}

void dictionary::accept()

{

string key,mean;

cout<<"Enter keyword "<<endl;

cin>>key;

cout<<"Enter meaning "<<endl;

cin>>mean;

insert(key,mean);

}

void dictionary::LLrotation(avlnode \*a,avlnode \*b)

{

cout<<"LL rotation"<<endl;

a->left=b->right;

b->right=a;

a->bf=b->bf=0;

}

void dictionary::RRrotation(avlnode \*a,avlnode \*b)

{

cout<<"RR rotation"<<endl;

a->right=b->left;

b->left=a;

a->bf=b->bf=0;

}

void dictionary::insert(string key,string mean)

{

//cout<<"IN Insert \n";

if(!root)

{

//create new root

root=new avlnode(key,mean);

cout<<"ROOT CREATED \n";

return;

}

// else

// {

avlnode \*a,\*pa,\*p,\*pp;

//a=NULL;

pa=NULL;

p=a=root;

pp=NULL;

while(p)

{

cout<<"In first while \n";

if(p->bf)

{

a=p;

pa=pp;

}

if(key<p->keyword){pp=p;p=p->left;} //takes the left branch

else if(key>p->keyword){pp=p;p=p->right;} //right branch

else

{

//p->meaning=mean;

cout<<"Already exist \n";

return;

}

}

cout<<"Outside while \n";

avlnode \*y=new avlnode(key,mean);

if(key<pp->keyword)

{

pp->left=y;

}

else

pp->right=y;

cout<<"KEY INSERTED \n";

int d;

avlnode \*b,\*c;

//a=pp;

b=c=NULL;

if(key>a->keyword)

{

cout<<"KEY >A->KEYWORD \n";

b=p=a->right;

d=-1;

cout<<" RIGHT HEAVY \n";

}

else

{

cout<<"KEY < A->KEYWORD \n";

b=p=a->left;

d=1;

cout<<" LEFT HEAVY \n";

}

while(p!=y)

{

if(key>p->keyword)

{

p->bf=-1;

p=p->right;

}

else

{

p->bf=1;

p=p->left;

}

}

cout<<" DONE ADJUSTING INTERMEDIATE NODES \n";

if(!(a->bf)||!(a->bf+d))

{

a->bf+=d;

return;

}

//else

//{

if(d==1)

{

//left heavy

if(b->bf==1)

{

LLrotation(a,b);

/\*a->left=b->right;

b->right=a;

a->bf=0;

b->bf=0;\*/

}

else //if(b->bf==-1)

{

cout<<"LR rotation"<<endl;

c=b->right;

b->right=c->left;

a->left=c->right;

c->left=b;

c->right=a;

switch(c->bf)

{

case 1:

{

a->bf=-1;

b->bf=0;

break;

}

case -1:

{

a->bf=0;

b->bf=1;

break;

}

case 0:

{

a->bf=0;

b->bf=0;

break;

}

}

c->bf=0;

b=c; //b is new root

}

//else

// cout<<"Balanced \n";

}

if(d==-1)

{

if(b->bf==-1)

{

// cout<<"RR rotation"<<endl;

/\*a->right=b->left;

b->left=a;

a->bf=b->bf=0;\*/

RRrotation(a,b);

}

else// if(b->bf==1)

{

c=b->left;

// cout<<"RL rotation"<<endl;

a->right=c->left;

b->left=c->right;

c->left=a;

c->right=b;

switch(c->bf)

{

case 1:

{

a->bf=0;

b->bf=-1;

break;

}

case -1:

{

a->bf=1;

b->bf=0;

break;

}

case 0:

{

a->bf=0;

b->bf=0;

break;

}

}

c->bf=0;

b=c; //b is new root

}

//else

//cout<<"Balanced \n";

}

//}

if(!pa)

root=b;

else if(a==pa->left)

pa->left=b;

else

pa->right=b;

cout<<"AVL tree created!! \n";

//cout<<"AVL \n";

//inorder(root);

}

void dictionary::search(string key)

{

cout<<"ENTER SEARCH \n";

loc=NULL;

par=NULL;

if(root==NULL)

{

cout<<"Tree not created "<<endl;

// root=key;

loc=NULL;

par=NULL;

}

//par=NULL;loc=NULL;

avlnode \*ptr;

ptr=root;

while(ptr!=NULL)

{

if(ptr->keyword==key)

{

//flag=1;

loc=ptr;

break; //imp for delete1 else it doesnt exit while loop

}

else if(key<ptr->keyword)

{

par=ptr;

ptr=ptr->left;

}

else

{

par=ptr; //edit this in previous code

ptr=ptr->right;

}

}

if(loc==NULL)

{

cout<<"Not found "<<endl;

}

}

void dictionary::update(string oldkey,string newmean)

{

search(oldkey);

loc->meaning=newmean;

cout<<"UPDATE SUCCESSFUL \n";

}

void dictionary::deletekey(string key)

{

}

void dictionary::inorder(avlnode \*root)

{

if(root)

{

inorder(root->left);

cout<<root->keyword<<" "<<root->meaning<<endl;

inorder(root->right);

}

}

int main()

{

string k,m;

dictionary d;

int ch;

string key,mean;

do

{

cout<<"1.Insert \n2.Update \n3.Ascending \n4.Descending \n5.Display \n6.Quit \n";

cin>>ch;

switch(ch)

{

case 1:

{

d.accept();

break;

}

case 2:

{

cout<<"Enter key whose meaning to update \n";

cin>>key;

cout<<"Enter new meaning\n";

cin>>mean;

d.update(key,mean);

break;

}

case 3:

d.inorder(d.root);

break;

case 4:

cout<<"Descending \n";

d.descending(d.root);

break;

case 5:

d.inorder(d.root);

break;

default:

break;

}

}while(ch!=6); /\*cout<<"Enter word and its meaning"<<endl;

cin>>k>>m;

d.insert(k,m);\*/

// d.accept();

//cout<<"Enter another word and its meaning \n";

// cin>>k>>m;

// d.insert(k,m);

//cout<<"MAIN \n";

return 0;

}