#include<iostream>

#include<cstring>

using namespace std;

struct node

{

char keyword[15],meaning[30];

struct node \*left,\*right;

int height;

};

class avldictionary //class

{

public:

struct node \*insertkeyword(struct node \*r,char ik[15],char im[15]);

struct node \*searchkeyword(struct node \*trav,char sk[15]);

int balanceFactor(struct node \*r);

int maxheight(struct node \*r);

struct node \*RR(struct node \*r);

struct node \*LL(struct node \*r);

struct node \*LR(struct node \*r);

struct node \*RL(struct node \*r);

void ascending(struct node \*r);

void descending(struct node \*r);

struct node \*del(struct node \*r,char k[15]);

};

int avldictionary::balanceFactor(struct node \*r) //return balance factor of node r

{

int lheight,rheight;

if(r->left==NULL)

lheight=0;

else

lheight=1+r->left->height;

if(r->right==NULL)

rheight=0;

else

rheight=1+r->right->height;

return(lheight-rheight); //return LST's and RST's height difference i.e. BF

}

int avldictionary::maxheight(struct node \*r) //return maxheight (either LST's or RST's)

{

int lheight,rheight;

if(r->left==NULL) //if r's LeftSubTree(LST) is NULL, height of LST is 0

lheight=0;

else

lheight=1+r->left->height;

if(r->right==NULL) //if r's RightSubTree(RST) is NULL, height of RST is 0

rheight=0;

else

rheight=1+r->right->height;

if(lheight > rheight)

return lheight;

else

return rheight;

}

struct node \*avldictionary::insertkeyword(struct node \*r,char ik[15],char im[15])

{

if(r==NULL)

{

r=new struct node;

strcpy(r->keyword,ik); //r's keyword and meaning

strcpy(r->meaning,im); //updated with values given by user

r->left=r->right=NULL; //r's both links are set to NULL

}

else if(strcmp(ik, r->keyword) > 0)

{

r->right=insertkeyword(r->right,ik,im);

if(balanceFactor(r)==-2) //BF is -2 then insertion in RightSubTree

{

if(strcmp(ik, r->right-> keyword) > 0)

r=LL(r); // if insertion in RST's RST then LL

else

r=RL(r); //else in RST's LST(Left Sub Tree) then RL

}

}

else if(strcmp(ik, r->keyword) < 0)

{

r->left=insertkeyword(r->left,ik,im);

if(balanceFactor(r)==2) //BF is 2 then isertion in LeftSubTree

{

if(strcmp(ik, r->left-> keyword) < 0)

r=RR(r); //if insertion in LST's LST then RR

else

r=LR(r); //else in LST's RST then LR

}

}

r->height=maxheight(r); //finds maxheight (either from LST or RST) of r

return r;

}

struct node \*avldictionary::RR(struct node \*parent) //RR rotation

{

struct node \*lchild;

lchild=parent->left;

parent->left=lchild->right;

lchild->right=parent;

parent->height=maxheight(parent);

lchild->height=maxheight(lchild);

return lchild;

}

struct node \*avldictionary::LL(struct node \*parent) //LL rotation

{

struct node \*rchild;

rchild=parent->right;

parent->right=rchild->left;

rchild->left=parent;

parent->height=maxheight(parent);

rchild->height=maxheight(rchild);

return rchild;

}

struct node \*avldictionary::LR(struct node \*parent) //LR double rotation

{

parent->left=LL(parent->left); //call single LL rotation

parent=RR(parent); //call single RR rotation

return parent;

}

struct node \*avldictionary::RL(struct node \*parent) //RL double rotation

{

parent->right=RR(parent->right); //call single RR rotation

parent=LL(parent); //call single LL roation

return parent;

}

void avldictionary::ascending(struct node \*r)

{

if(r!=NULL)

{

ascending(r->left);

cout.width(15);

cout<<r->keyword;

cout<<"|";

cout.width(30);

cout<<r->meaning;

cout<<"|";

cout<<"\n-----------------------------------------------\n";

ascending(r->right);

}

}

void avldictionary::descending(struct node \*r)

{

if(r!=NULL)

{

descending(r->right);

cout.width(15);

cout<<r->keyword;

cout<<"|";

cout.width(30);

cout<<r->meaning;

cout<<"|";

cout<<"\n-----------------------------------------------\n";

descending(r->left);

}

}

struct node \*avldictionary::searchkeyword(struct node \*trav,char sk[15])

{ //func. to search keyword in BST

int count=0;

while(trav!=NULL)

{

count++; //counts no. of comparision needed

if(strcmp(sk,trav->keyword)==0)

{

cout<<"\n\n Keyword FOUND Successfullly...!"; //keyword found

cout<<"\n No. of comparions required are: "<<count;

return trav;

}

else if(strcmp(sk,trav->keyword)>0)

{

trav=trav->right; //traverse to right subtree

}

else

{

trav=trav->left; //traverse to left subtree

}

}

return trav; //return trav=NULL when Keyword not found,

//return trav=BST node matched with given keyword

}

struct node \* avldictionary::del(struct node \*r,char k[15])

{

node \*temp;

if(r==NULL)

return NULL;

else

{

if(strcmp(r->keyword,k)<0)

{

r->right=del(r->right,k);

if(balanceFactor(r)==2)

{

if(balanceFactor(r->left)>=0)

r=LL(r);

else

r=LR(r);

}

}

else if(strcmp(r->keyword,k)>0)

{

r->left=del(r->left,k);

if(balanceFactor(r)==-2)

{

if(balanceFactor(r->right)<=0)

r=RR(r);

else

r=RL(r);

}

}

else//Data to be Deleted is found

{

if(r->right!=NULL)

{

temp=r->right;

while(temp->left!=NULL)

temp=temp->left;

strcpy(r->keyword,temp->keyword);

strcpy(r->meaning,temp->meaning);

r->right=del(r->right,temp->keyword);

if(balanceFactor(r)==2)

{

if(balanceFactor(r->left)>=0)

r=LL(r);

else

r=LR(r);

}

}

else

return(r->left);

}

r->height=maxheight(r);

return r;

}

}

int main()

{

char k[15],m[30];

int choice,n;

struct node \*root=NULL,\*found=NULL; //create root pointer and set to NULL

avldictionary obj; //object of dictionary class created

do{

cout<<endl;

cout<<"1. ENTER NEW KEYWORD."<<endl;

cout<<"2. SEARCH KEYWORD."<<endl;

cout<<"3. PRINT DICTIONARY ASCENDING ORDER."<<endl;

cout<<"4. PRINT DICTIONARY DESCENDING ORDER."<<endl;

cout<<"5. DELETE."<<endl;

cout<<"6. UPDATE THE MEANING OF KEYWORD."<<endl;

cout<<"7. EXIT."<<endl;

cout<<" Enter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\n How many keyword you want to insert: ";

cin>>n;

cin.getline(k,0);

for(int i=0;i<n;i++) //loop to accept n keywords and meaning

{

cout<<"\n Enter keyword: ";

cin.getline(k,15);

cout<<" Enter meaning: ";

cin.getline(m,30);

root=obj.insertkeyword(root,k,m); //inserts keywords to BST

}

cout<<"\n Keyword inserted Successfully....!\n";

break;

case 2:

cout<<"\n Enter keyword to be searched: ";

cin>>k;

found=obj.searchkeyword(root,k); //function call to search

//keyword k in BST

if(found==NULL)

{

cout<<"\n Keyword NOT present...\n";

}

else

{ //if 'found' is not NULL it contains

cout<<endl<<endl<<" "; //BST node searched in BST

cout<<found->keyword<<"==>"; //print information of 'found'

cout<<found->meaning;

cout<<endl;

}

break;

case 3:

cout<<"\n Keywords in Ascending Order\n";

cout<<"\n \n";

cout.width(15);

cout<<"Dict. Keyword"; cout<<"|";

cout.width(30);

cout<<"Keyword's Meaning";

cout<<"|\n";

cout<<"===============================================\n";

obj.ascending(root); //prints dictionary in ascending order

break;

case 4:

cout<<"\n Descending Order\n";

cout<<"\n \n";

cout.width(15);

cout<<"Dict. Keyword";

cout<<"|";

cout.width(30);

cout<<"Keyword's Meaning";

cout<<"|\n";

cout<<"===============================================\n";

obj.descending(root);

cout<<"\n Dictonary Printed Successfully....!\n";

break;

case 5:

cout<<"Enter rhe Keyword to be Deleted";

cin.getline(k,0);

cin.getline(k,15);

root=obj.del(root,k);

break;

case 6:

cout<<"Enter the Keyword Whose meaning needs to be Updated";

cin.getline(k,0);

cin.getline(k,15);

found=obj.searchkeyword(root,k);

if(found==NULL)

cout<<"No such Keyword present to update meaning";

else

{

cout<<"Enter the new Meaning";

cin.getline(m,30);

strcpy(found->meaning,m);

cout<<"Keyword's Meaning is updated successfully";

}

}//switch ends...

}while(choice!=7);

return 0;

}