```
/*
Author: Parag Ghorpade

PROBLEM STATEMENT:

Dictionary  stores  keywords  &  its  meanings.  Provide  facility  for  adding  new
 keywords, deleting keywords, updating values of any entry.
Also,display whole data sorted in ascending or descending order.
Find how many maximum comparisons may require for finding any keyword. Use height
balance tree and find the complexity for finding a keyword.

*/


#include<iostream>
#include<string.h>
using namespace std;
class dict
{
    dict *root,*node,*left,*right,*tree1;
    string s1,s2;
    int flag,flag1,flag2,flag3,cmp;
public:
    dict()
    {
        flag=0,flag1=0,flag2=0,flag3=0,cmp=0;
        root=NULL;
    }
    void input();
    void create_root(dict*,dict*);
    void check_same(dict*,dict*);
    void input_display();
    void display(dict*);
    void input_remove();
    dict* remove(dict*,string);
    dict* findmin(dict*);
    void input_find();
    dict* find(dict*,string);
    void input_update();
    dict* update(dict*,string);

};

                void dict::input()
                {
                           node=new dict;
                           cout<<"\nEnter the keyword:\n";
                           cin>>node->s1;
                           cout<<"Enter the meaning of the keyword:\n";
                           cin.ignore();
```

```
                        getline(cin,node->s2);
                        create_root(root,node);
                }


                                void dict::create_root(dict *tree,dict *node1)
                                {
                                    int i=0,result;
                                    char a[20],b[20];
                                    if(root==NULL)
                                    {
                                        root=new dict;
                                        root=node1;
                                        root->left=NULL;
                                        root->right=NULL;
                                        cout<<"\nRoot node created
successfully"<<endl;

                                        return;
                                    }
                                    for(i=0;node1->s1[i]!='\0';i++)
                                    {
                                        a[i]=node1->s1[i];
                                    }
                                    for(i=0;tree->s1[i]!='\0';i++)
                                    {
                                        b[i]=tree->s1[i];
                                    }
                                    result=strcmp(b,a);
                                    check_same(tree,node1);
                                    if(flag==1)
                                        {
                                            cout<<"The word you entered already
exists.\n";

                                            flag=0;
                                        }
                                        else
                                        {
                                    if(result>0)
                                    {
                                        if(tree->left!=NULL)
                                        {
                                            create_root(tree->left,node1);
                                        }
                                        else
                                        {
                                            tree->left=node1;
                                            (tree->left)->left=NULL;
                                            (tree->left)->right=NULL;
                                        cout<<"Node added to left of
"<<tree->s1<<"\n";
```

```
                                        return;
                                       }
                                    }
                                    else if(result<0)
                                    {
                                         if(tree->right!=NULL)
                                         {
                                             create_root(tree->right,node1);
                                         }
                                         else
                                         {
                                             tree->right=node1;
                                             (tree->right)->left=NULL;
                                             (tree->right)->right=NULL;
                                             cout<<"Node added to right of 
"<<tree->s1<<"\n";

                                             return;
                                         }
                                    }
                                         }
                                    }


void dict::check_same(dict *tree,dict *node1)
{
        if(tree->s1==node1->s1)
        {
                flag=1;
                return;
        }
        else if(tree->s1>node1->s1)
     {
         if(tree->left!=NULL)
        {
         check_same(tree->left,node1);
        }
     }
     else if(tree->s1<node1->s1)
     {
         if(tree->right!=NULL)
         {
         check_same(tree->right,node1);
         }
     }
}


                void dict::input_display()
                {
                        if(root!=NULL)
                        {
                        cout<<"The words entered in the dictionary are:\n\n";
                        display(root);
                        }
                        else
                        {
                        cout<<"\nThere are no words in the dictionary.\n";
                        }
                }


                                void dict::display(dict *tree)
                                {
                                        if(tree->left==NULL&&tree->right==NULL)
                                        {
                                                cout<<tree->s1<<" = 
"<<tree->s2<<"\n\n";
                                        }
                                        else
                                        {
                                        if(tree->left!=NULL)
                                        {
                                                display(tree->left);
                                        }
                                        cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
                                        if(tree->right!=NULL)
                                        {
                                                display(tree->right);
                                        }
                                        }
                                }


void dict::input_remove()
{
        char t;
        if(root!=NULL)
        {
          cout<<"\nEnter a keyword to be deleted:\n";
          cin>>s1;
          remove(root,s1);
          if(flag1==0)
          {
                cout<<"\nThe word '"<<s1<<"' has been deleted.\n";
          }
          flag1=0;
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
        }
}


                dict* dict::remove(dict *tree,string s3)
                {
                        dict *temp;
                    if(tree==NULL)
                    {
                        cout<<"\nWord not found.\n";
                        flag1=1;
                        return tree;
                    }
                    else if(tree->s1>s3)
                    {
                        tree->left=remove(tree->left,s3);
                        return tree;
                    }
                    else if(tree->s1<s3)
                            {
                        tree->right=remove(tree->right,s3);
                        return tree;
                    }
                    else
                    {
                        if(tree->left==NULL&&tree->right==NULL)
                        {
                                delete tree;
                                tree=NULL;
                        }
                        else if(tree->left==NULL)
                        {
                                temp=tree;
                                tree=tree->right;
                                delete temp;
                        }
                        else if(tree->right==NULL)
                        {
                                temp=tree;
                                tree=tree->left;
                                delete temp;
                        }
                        else
                        {
                                temp=findmin(tree->right);
                                tree=temp;
                                tree->right=remove(tree->right,temp->s1);
                        }
                }
                return tree;
```

```
}


                dict* dict::findmin(dict *tree)
                {
                        while(tree->left!=NULL)
                        {
                                tree=tree->left;
                        }
                        return tree;
                }


void dict::input_find()
{
        flag2=0,cmp=0;
        if(root!=NULL)
        {
        cout<<"\nEnter the keyword to be searched:\n";
        cin>>s1;
    find(root,s1);
    if(flag2==0)
    {
                cout<<"Number of comparisons needed: "<<cmp<<"\n";
                cmp=0;
    }
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
        }
}


                dict* dict::find(dict *tree,string s3)
                {
                        if(tree==NULL)
                        {
                                cout<<"\nWord not found.\n";
                                flag2=1;
                                flag3=1;
                                cmp=0;
                        }
                        else
                        {
                                if(tree->s1==s3)
                                {
                                        cmp++;
                                        cout<<"\nWord found.\n";
                                        cout<<tree->s1<<":
```

```
"<<tree->s2<<"\n";
                                                                tree1=tree;
                                                                return tree;
                                                        }
                                                        else if(tree->s1>s3)
                                                        {
                                                                cmp++;
                                                                find(tree->left,s3);
                                                        }
                                                        else if(tree->s1<s3)
                                                        {
                                                                cmp++;
                                                                find(tree->right,s3);
                                                        }
                                                }
                                                return tree;
                                        }


void dict::input_update()
{
        if(root!=NULL)
        {
        cout<<"\nEnter the keyword to be updated:\n";
        cin>>s1;
    update(root,s1);
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
        }
}


                dict* dict::update(dict *tree,string s3)
                {
                        flag3=0;
                        find(tree,s3);
                        if(flag3==0)
                        {
                    cout<<"\nEnter the updated meaning of the keyword:\n";
                    cin.ignore();
                    getline(cin,tree1->s2);
                    cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";
                        }
                    return tree;
                }
```

```
                                int main()
                                  {
                                    int ch;
                                    dict d;
                                    do
                                    {

cout<<"\n==========================================\n"
                                          "\n********DICTIONARY***********:\n"
                                         "\nEnter your choice:\n"
                                         "1.Add new keyword.\n"
                                         "2.Display the contents of the
Dictionary.\n"
                                         "3.Delete a keyword.\n"
                                         "4.Find a keyword.\n"
                                         "5.Update the meaning of a keyword.\n"
                                         "6.Exit.\n"

"===============================================\n";
                                cin>>ch;
                                switch(ch)
                                {
                                    case 1:d.input();
                                           break;
                                    case 2:d.input_display();
                                           break;
                                    case 3:d.input_remove();
                                           break;
                                    case 4:d.input_find();
                                           break;
                                    case 5:d.input_update();
                                           break;
                                    default:cout<<"\nPlease enter a valid
option!\n";
                                            break;
                                }
                                  }while(ch!=6);
                                return 0;
                                }
```