```
/*
A Dictionary stores keywords &amp;amp; its meanings. Provide facility for adding new keywords,
deleting keywords, &amp;amp; updating values of any entry. Also provide facility to display whole
data sorted in ascending/ Descending order, Also find how many maximum comparisons may require
for finding any keyword. Make use of appropriate data structures.
*/
#include<iostream>
#include<string.h>
using namespace std;

class node
{
    public: char key[20];
        char meaning[20];
        node *left,*right;

        node()
        {
            left=right=NULL;
        }
};

class dictionary
{
    public:
    node *root;
        dictionary()
        {
            root=NULL;
        }
        void addkey();
        void insert(node*);
        void display_asc();
        void inorder(node*);
        void display_desc();
        void rtorder(node*);
        void findkey();
        void updatekey();
        void deletekey();
        node* bstsearch(node *,char[]);
        void bstdelete(node*,node*);
        node* parent(node*,node*);
        node *findlargest(node*);
};

void dictionary::addkey()
{
    node *temp;
    int n,i;

    cout<<"\nEnter no of keys to be inserted: ";
    cin>>n;

    for(i=0;i<n;i++)
```

```
    {
        temp=new node();
        cout<<"\nEnter Keyword : ";
        cin>>temp->key;
        cout<<"\nEnter Meaning : ";
        cin>>temp->meaning;
        insert(temp);
    }
}

void dictionary::display_asc()
{
    if(root==NULL)
    cout<<"\nDictionary is Empty";
    else
    inorder(root);
}
void dictionary::display_desc()
{
    if(root==NULL)
    cout<<"\nDictionary is Empty";
    else
    rtorder(root);
}
void dictionary::inorder(node *r)
{
    if(r!=NULL)
    {
        inorder(r->left);
        cout<<"\n"<<r->key<<" : =  "<<r->meaning;
        inorder(r->right);
    }
}
void dictionary::rtorder(node *r)
{
    if(r!=NULL)
    {
        rtorder(r->right);
        cout<<"\n"<<r->key<<" : =  "<<r->meaning;
        rtorder(r->left);
    }
}

void dictionary::insert(node *temp)
{
    node *temp1,*temp2;

    if(root==NULL)
    {
        root=temp;
    }
    else
    {
        temp1=root;
```

```
        while(temp1!=NULL)
        {
            temp2=temp1;
            if(strcmp(temp->key,temp1->key)<0)
            temp1=temp1->left;
            else
            temp1=temp1->right;
        }
        if(strcmp(temp->key,temp2->key)<0)
        temp2->left=temp;
        else
        temp2->right=temp;
    }
}

void dictionary::findkey()
{
    char key[20];
    node *temp;

    cout<<"\nEnter key to be searched: ";
    cin>>key;
    temp=bstsearch(root,key);
    if(temp==NULL)
    cout<<"\nKey is not present in the Dictionary";
    else
    {       cout<<"\nKey Found..!!!";
        cout<<"\n"<<temp->key<<" : "<<temp->meaning;
    }
}
void dictionary::updatekey()
{
    char key[20];
    node *temp;

    cout<<"\nEnter key to be Updated: ";
    cin>>key;
    temp=bstsearch(root,key);
    if(temp==NULL)
    cout<<"\nKey is not present in the Dictionary";
    else
    {       cout<<"\nEnter New meaning for "<<temp->key;
        cin>>temp->meaning;
        cout<<"\n"<<temp->key<<" : "<<temp->meaning;
    }
}

void dictionary::deletekey()
{
    char key[20];
    node *temp1,*temp2,*largest;
    cout<<"\nEnter key to be deleted: ";
    cin>>key;
    temp1=bstsearch(root,key);
```

```
    if(temp1==NULL)
    cout<<"\nKey is not Present in dictionary";
    else
    {
    cout<<"\nKey deleted Successfully..!!";
    if(temp1->left==NULL&&temp1==root)
    {
        root=temp1->right;
        delete temp1;
    }
    else if(temp1==root&&temp1->left!=NULL)
    {
        largest=findlargest(temp1->left);
        strcpy(temp1->key,largest->key);
        strcpy(temp1->meaning,largest->meaning);
        if(temp1->left==largest)
        {
            temp1->left=NULL;
        }
        else
        {
            temp2=parent(temp1->left,largest);
            bstdelete(largest,temp2);
        }
    }
    else
    {
        temp2=parent(root,temp1);
        bstdelete(temp1,temp2);
    }
    }
}
void dictionary::bstdelete(node *temp1,node *temp2)
{
    node *largest;

    if(strcmp(temp1->key,temp2->key)<0)
    {
        if(temp1->left==NULL)
        {
            temp2->left=temp1->right;
            delete temp1;
        }
        else if(temp1->right==NULL)
        {
            temp2->left=temp1->right;
            delete temp1;
        }
        else
        {
            largest=findlargest(temp1->left);
            strcpy(temp1->key,largest->key);
            strcpy(temp1->meaning,largest->meaning);
            temp2=parent(temp1->left,largest);
```

```
            bstdelete(largest,temp2);
        }

    }
    else
    {
        if(temp1->left==NULL)
        {
            temp2->right=temp1->right;
            delete temp1;
        }
        else if(temp1->right==NULL)
        {
            temp2->right=temp1->right;
            delete temp1;
        }
        else
        {
            largest=findlargest(temp1->left);
            strcpy(temp1->key,largest->key);
            strcpy(temp1->meaning,largest->meaning);
            temp2=parent(temp1->left,largest);
            bstdelete(largest,temp2);
        }

    }
}
node *dictionary::parent(node *r,node*temp)
{
    node *temp1,*temp2;
    temp1=r;
    while(temp1!=temp)
    {
        temp2=temp1;
        if(strcmp(temp->key,temp1->key)<0)
        temp1=temp1->left;
        else
        temp1=temp1->right;
    }
    return temp2;
}
node *dictionary::findlargest(node *t)
{
    if(t->right==NULL)
    return t;
    else
    return findlargest(t->right);
}

node *dictionary::bstsearch(node *r,char key[20])
{
    if(r==NULL)
    return NULL;
    if(strcmp(key,r->key)<0)
```

```
    return bstsearch(r->left,key);
    else if(strcmp(key,r->key)>0)
    return bstsearch(r->right,key);
    else
    return r;
}
int main()
{
    int choice;
    char ch;
    dictionary dct;
    do
    {
        cout<<"****** MENU *****";
        cout<<"\n1. Add keys to Dictionary";
        cout<<"\n2. Display Dictionary in Ascending order";
        cout<<"\n3. Display Dictionary in Descending order";
        cout<<"\n4. Find Key";
        cout<<"\n5. Update Key";
        cout<<"\n6. Delete Key";
        cout<<"\n\nEnter your choice: ";
        cin>>choice;
        switch(choice)
        {
            case 1: dct.addkey();
                break;
            case 2: dct.display_asc();
                break;
            case 3: dct.display_desc();
                break;
            case 4: dct.findkey();
                break;
            case 5: dct.updatekey();
                break;
            case 6: dct.deletekey();
                break;
            default:cout<<"\n\nInvalid Choie....!!!";
        }
        cout<<"\n\nDo you want to continue..[y/n] ? : ";
        cin>>ch;
    }while(ch!='n');
    return 0;
}
```