```
#include <iostream>
#include<stdlib.h>
using namespace std;

int count;
class BTNODE
{
public:
      string word;
      string mean;
      BTNODE *left;
      BTNODE *right;
      BTNODE()
      {
        left=right=NULL;
        word=mean=-1;
      }

};
class DICTIONARY
{
public:
      BTNODE *root;
      DICTIONARY()
     {
         root=NULL;
      }

      void create();
      BTNODE *search(string);
      void Insert(string,string);
     void InOrder(BTNODE *);
      BTNODE *getBTNODE(string,string);
     BTNODE *findparent(BTNODE *);
     void delet1(BTNODE *);
};
//function for searching word in dictionary
BTNODE *DICTIONARY::search(string key)
{
      BTNODE *T=root;
      while(T!=NULL)
      {
            if(T->word==key )
            {
                  count++;
                  return T;
            }
            else if(T->word>key)
            {
                  count++;
              T=T->left;
            }
```

```
            else
            {
                  count++;
                  T=T->right;
            }

      }
      return T;
}
//function for creating node in dictionary
BTNODE *DICTIONARY::getBTNODE(string key,string min)
{
      BTNODE *p;
      p=new BTNODE;
      if(!p)
      {
            cout<<"memory is not sufficient"<<endl;
      }
      else
      {
            p->word=key;
            p->mean=min;
      }
      return p;
}

//function for insert word in dictionary
void DICTIONARY::Insert(string key,string min)
{
  BTNODE *p=getBTNODE(key,min);
  if(root==NULL)
  {
       root=p;
  }
  else
  {
        BTNODE *T=root;
        while(1)
        {
              if(key<T->word)
              {
                    if(T->left==NULL)
                    {
                          T->left=p;
                          break;
                    }
                    else
                    {
                          T=T->left;
                    }
              }
              else
```

```
            {
                if(T->right==NULL)
                {
                      T->right=p;
                      break;
                }
                else
                {
                      T=T->right;
                }
            }
        }
  }
}
//function for the dictionary
void DICTIONARY::create()
{
        string key,min;
            char ch;
            do
            {
                  cout<<"\n enter the word =>";
                  cin>>key;
                  cout<<"\n enter the meaning =>";
                  cin>>min;
                  if(search(key)==NULL)
                  {

                        Insert(key,min);
                  }
                  else
                  {
                        cout<<"\n word is already present.....";
                  }

                  cout<<"\n do you want to add more word=>";
                  cin>>ch;


            }while(ch=='y'||ch=='Y');
}
//function for displaying dictionary

void DICTIONARY::InOrder(BTNODE *T)
{
      if(T!=NULL)
      {
            InOrder(T->left);
            cout<<endl<<T->word<<"-:"<<T->mean;
            InOrder(T->right);
      }
}
```

```
//function for finding parent

BTNODE *DICTIONARY::findparent(BTNODE *K)
{
      BTNODE *q=root;
      if(K==root)
      {
            return NULL;
      }
      else
      {
            while(1)
       {
            if((q->left==K)||(q->right==K))
            {
                  return q;
            }
            else
            {
                  if(K->word<q->word)
                  {
                        q=q->left;
                  }
                  else
                  {
                        q=q->right;
                  }
            }

       }

      }
      return NULL;
}

// function for deleting word from DICTONARY.

void DICTIONARY::delet1(BTNODE *q)
{

      BTNODE *p;
      if((q->left==NULL)&&(q->right==NULL))
      {
            if(q==root)
            {
                  root=NULL;
                  delete q;
                  return;
            }
         else
        {
            p=findparent(q);
```

```
            if(p->left==q)
            {
                  p->left=NULL;
                  delete q;
                  return;
            }
            else
            {
                  p->right=NULL;
                  delete q;
                  return;
            }
         }
    }
            //logic for deleting leaf node.
            if((q->left==NULL && q->right!=NULL) || (q->left!=NULL &&
q->right==NULL))
            {

                  if(q==root)
                  {
                        if(q->left!=NULL)
                        {
                              root=q->left;
                              delete q;
                              return;
                        }
                        else
                        {
                              root=q->right;
                              delete q;
                              return;
                        }
                  }
                  else
                  {
                        p=findparent(q);
                        if(p->left==q)
                        {
                           if(q->left!=NULL)
                           {
                                   p->left=q->left;
                                   delete q;
                                   return;
                           }
                           else
                              {
                                   p->left=q->right;
                                   delete q;
                                   return;
                              }
                        }
```

```
                        else
                        {
                              if(q->left!=NULL)
                              {
                                    p->right=q->left;
                                    delete q;
                                    return;
                              }
                              else
                              {
                                    p->right=q->right;
                                    delete q;
                                    return;
                              }
                        }

                  }
            }  // logic for deleting 1 degree node.

            BTNODE *S=q->right;
            while(S->left!=NULL)
            {
                  S=S->left;
            }

            q->word=S->word;
            delet1(S);
            // logic for deleting 2 degree node.
}



int main() {
         string key;
         DICTIONARY b;
           int ch;
           while(ch!=6)
      {

            cout<<"\n 1) create dictonary";
            cout<<"\n 2) display dictonary using inorder sequence";
            cout<<"\n 3) serach word meaning in the dictonary";
            cout<<"\n 4) insert word in the dictonary";
            cout<<"\n 5) delete the word";
            cout<<"\n 6) exit.";
            cout<<"\n enter choice.....=>";
            cin>>ch;
            switch(ch)
            {
            case 1:b.create();
                   break;
            case 2:cout<<"\n DICTIONARY is =>"<<endl;
```

```
                    b.InOrder(b.root);
                    break;
           case 3:cout<<"\n enter word whose meaning to find =>";
                  cin>>key;
                  count=0;
                    if(b.search(key)==NULL)
                    {
                   cout<<"\n word is not found....";

                   }
                    else
                  {
                     cout<<"\n "<<b.search(key)->word<<"-
:"<<b.search(key)->mean<<" required comparisons="<<count;
                  }

                  break;
        case 4:b.create();
               cout<<"\n after insertion DICTIONARY is =>"<<endl;
                   b.InOrder(b.root);
                   break;
        case 5:cout<<"\n enter word which you want  to delete =>";
               cin>>key;
               BTNODE *p=b.search(key);
               b.delet1(p);
               cout<<"\n after deletion DICTIONARY becomes"<<endl;
               b.InOrder(b.root);
               break;




           }

           }
     return 0;
}
```