==========

Roll Number: SYCOC303 Division: C

PRN Number: 122B2B303 Batch: C4

Name: VINAYAK MADAN SHETE

==========

Problem Statement:

- **A Dictionary stores keywords & its meaning. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order.Also find how many maximum comparisons may require for finding any keyword. Use Binary SearchTree for implementation.**

==========

INPUT:

```
/*
 *     ====================================
 *           Program Name: DictionaryUingBST.cpp
 *     Created on: December 28, 2022
 *      Author: Vinayak Shete
 *  ====================================
 */

#include <iostream>
#include<string>
using namespace std;
class dictionary;
class node
{
        string word,meaning;
        node *left,*right;
       public:
              friend class dictionary;
              node()
              {
```

```
                left=NULL;
                right=NULL;
        }
        node(string word, string meaning)
        {
         this->word=word;
         this->meaning=meaning;
         left=NULL;
         right=NULL;
        }
};

class dictionary
{
        node *root;
        public:
                dictionary()
                {
                        root=NULL;
                }
                void create();
                void inorder_rec(node *rnode);
                void postorder_rec(node *rnode);
                //inorder traversal of BST for ascending order
                void inorder()
                {
                        inorder_rec(root);
                }
                void postorder();

                bool insert(string word,string meaning);
                int search(string key);

};

//function for searching a particular word
int dictionary::search(string key)
```

```
{
        node *tmp=root;
        int count;
        if(tmp==NULL)
        {
              return -1;
        }
        if(root->word==key)
         return 1;
        while(tmp!=NULL)
        {
                if((tmp->word)>key)
                {
                 tmp=tmp->left;
                 count++;
                }
                else if((tmp->word)<key)
                {
                 tmp=tmp->right;
                 count++;
                }
                else if(tmp->word==key)
                {
                 return ++count;
                }
        }
        return -1;

}

//postorder traversal of BST for descending order
void dictionary::postorder()
{
       postorder_rec(root);
}

void dictionary::postorder_rec(node *rnode)
```

```
{
        if(rnode)
        {
         postorder_rec(rnode->right);
         cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
         postorder_rec(rnode->left);
        }
}

//creating BST
void dictionary::create()
{
        int n;
        string wordI,meaningI;
        cout<<"\nHow many Words to insert?:";
        cin>>n;
        for(int i=0;i<n;i++)
        {
              cout<<"\n==================";
              cout<<"\nEnter Word "<<i+1<<": ";
              cin>>wordI;
              cout<<"\nEnter its Meaning: ";
              cin>>meaningI;
              insert(wordI,meaningI);
              cout<<"\n==================";
        }
}

void dictionary::inorder_rec(node *rnode)
{
        if(rnode)
        {
         inorder_rec(rnode->left);
         cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
         inorder_rec(rnode->right);
        }
}
```

```
//adding new word
bool dictionary::insert(string word, string meaning)
{
        node *p=new node(word, meaning);
        if(root==NULL)
        {
         root=p;
         return true;
        }
        node *cur=root;
        node *par=root;
        while(cur!=NULL) //traversal
        {
         if(word>cur->word)
         {par=cur;
         cur=cur->right;
         }
         else if(word<cur->word)
         {
          par=cur;
          cur=cur->left;
         }
         else
         {
          cout<<"\nWord is already present in the dictionary.";
          return false;
         }
        }
        if(word>par->word) //insertion of node
        {
         par->right=p;
         return true;
        }
        else
        {
         par->left=p;
```

```
         return true;
        }
}

int main()
{
        string word;
        dictionary months;
        cout<<"\n===============WELCOME============";
        months.create();
        cout<<"\nAscending order\n";
        months.inorder();

        cout<<"\nDescending order:\n";
        months.postorder();

        cout<<"\nEnter word to search: ";
        cin>>word;
        int comparisons=months.search(word);
        if(comparisons==-1)
        {
         cout<<"\nNot found word";
        }
        else
        {
         cout<<"\n "<<word<<" found in "<<comparisons<<" comparisons";
        }
        cout<<"\n===============THANK YOU!============";
        return 0;
}
```

==========

OUTPUT:

```
===============WELCOME============
How many Words to insert?:5

==================
Enter Word 1: Ambigue

Enter its Meaning: AmbiguousExpression

==================
==================
Enter Word 2: Computer

Enter its Meaning: Machine

==================
==================
Enter Word 3: Cringe

Enter its Meaning: Embarassed

==================
==================
Enter Word 4: Gratuitos

Enter its Meaning: Unwarranted

==================
==================
Enter Word 5: Galvanize

Enter its Meaning: StimulateAction

==================
```

```
==================
Ascending order
 Ambigue : AmbiguousExpression
 Computer : Machine
 Cringe : Embarassed
 Galvanize : StimulateAction
 Gratuitos : Unwarranted

Descending order:
 Gratuitos : Unwarranted
 Galvanize : StimulateAction
 Cringe : Embarassed
 Computer : Machine
 Ambigue : AmbiguousExpression

Enter word to search: Cringe

 Cringe found in 3 comparisons
==============THANK YOU!============
--------------------------------
Process exited after 131.6 seconds with return value 0
Press any key to continue . . .
```

==========