```
//============================================================================
========
// Name        : 21286_Assignment2.cpp
// Author      : Shrikrushna S Zirape
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
// Problem Sta : A Dictionary stores keywords and its meanings. Provide facility for adding
//                          new keywords, deleting keywords, updating values of any entry.
//                          Provide facility to display whole data sorted in ascending/
Descending order.
//                          Also find how many maximum comparisons may require for
finding any keyword.
//                          Use Binary Search Tree for implementation.
//============================================================================
========

#include <iostream>
#include <string.h>
using namespace std;

class Node{
        Node *lchild;
        Node *rchild;
        string word;
        string meaning;
public:
        Node(){
                lchild = NULL;
                rchild = NULL;
```

```
        }

        Node(string word, string meaning){
                this->word = word;
                this->meaning = meaning;
                this->lchild =NULL;
                this->rchild = NULL;
        }
        friend class BST;
};


class BST{
        Node * root;
        Node * parent;
public:
        BST(){
                root =NULL;
                parent = NULL;
        }
        Node * getRoot();
        Node * search(string s);
        void  insert(string word, string mean);
        Node * inorderSuccessor(Node *);
        void   deleteNode(string word);
        void   deleteLeafNode(string s);

        void ascendingDisplay(Node * temp);
```

```
        void descendingDisplay(Node *temp);

        void updateNode(string word, string meaning);
};

Node * BST::getRoot(){
        return root;
}

Node * BST::search(string s){
        int comparisons = 0;
                parent = NULL;
                Node* temp = root;
                if(temp == NULL) {
                        comparisons++;
                        return NULL;
                }

                while(temp != NULL){
                        if(s > temp->word){
                                parent = temp;
                                temp = temp->rchild;
                                comparisons++;
                        }
                        else if(s < temp->word){
                                parent = temp;
                                temp = temp->lchild;
                                comparisons++;
```

```
                }
                else if(s == temp->word){
                        comparisons++;
                        return temp;
                }
        }
        cout<<"\nNumber of comparisons are: "<<comparisons;
        return NULL;

}

void BST :: deleteNode(string word){
        Node* n = search(word);
        if(n == NULL){
                cout<<"\nNot found ....";
                return;
        }

        if(n->lchild == NULL && n->rchild == NULL){
                deleteLeafNode(word);
                return;
        }

        if(n->lchild == NULL || n->rchild == NULL){
                if(n->lchild == NULL){
                        if(parent->word > n->word){
                                parent->lchild = n->rchild;
                                delete(n);
```

```
                return;
            }

            if(parent->word < n->word){
                parent->rchild = n->rchild;
                delete(n);
                return;
            }
        }

        if(n->rchild == NULL){
            if(parent->word > n->word){
                parent->lchild = n->lchild;
                delete(n);
                return;
            }

            if(parent->word < n->word){
                parent->rchild = n->lchild;
                delete(n);
                return;
            }
        }
    }

    else{
        Node* temp = inorderSuccessor(n->rchild);
        n->word = temp->word;
```

```
            n->meaning = temp->meaning;
            deleteNode(temp->word);
        }
}

void BST::deleteLeafNode(string s){
        Node* n = this->search(s);
        if (n == NULL) {
            cout<<"does not exist";
            return;
        }

        if(parent->word > n->word){
            parent->lchild = NULL;
            delete(n);
            return;
        }

        if(parent->word < n->word){
            parent->rchild = NULL;
            delete(n);
            return;
        }
}

Node* BST :: inorderSuccessor(Node* temp){
        while(temp->lchild != NULL) temp = temp->lchild;
        return temp;
```

```
}

void BST::insert(string word, string meaning){
	Node* n;
	n = search(word);
	if(root == NULL && n == NULL){
		root = new Node(word, meaning);
		return;
	}

	if(n == NULL){
		if(word > parent->word){
			parent->rchild = new Node(word, meaning);
			return;
		}
		else{
			parent->lchild = new Node(word, meaning);
			return;
		}
	}

	cout<<"\nWord already exists";
	return;
}

void BST::ascendingDisplay(Node *temp){
	cout<<"\n";
	if(temp != NULL){
```

```
            ascendingDisplay(temp->lchild);
            cout<<temp->word<<" : "<<temp->meaning;
            ascendingDisplay(temp->rchild);
        }
}

void BST::descendingDisplay(Node *temp){
        cout<<"\n";
        if(temp != NULL){
            descendingDisplay(temp->rchild);
            cout<<temp->word<<" : "<<temp->meaning;
            descendingDisplay(temp->lchild);
        }
}

void BST::updateNode(string s, string m){
        Node *n = search(s);
        if (n==NULL){
            cout<<"Keyword does not exist.";
            return;
        }
        n->meaning = m;
}


int main() {
        BST b;
        int ch;
```

```
        string word, meaning;
//      b.insert("shri", "shrikrushna");
//      b.insert("zir", "zirape");
//      b.ascendingDisplay(b.getRoot());
//      b.descendingDisplay(b.getRoot());
        do{
                cout<<"\n----------Menu----------\n";
                cout<<"\n1. Insert";
                cout<<"\n2. Search";
                cout<<"\n3. print Ascending";
                cout<<"\n4. Print Descending";
                cout<<"\n5. Update";
                cout<<"\n6. Delete";
                cout<<"\n0. End the program";
                cout<<"\n Enter your Choice";
                cout<<"\n-----------------------------------------";
                cin>>ch;
                switch(ch){
                        case 1:
                                cout<<"\nEnter word : ";
                                cin>>word;
                                cout<<"\nEnter meaning : ";
                                cin>>meaning;
                                b.insert(word, meaning);
                                break;

                        case 2:
                                cout<<"\nEnter word to be searched: ";
```

```
            cin>>word;
            b.search(word);
            break;

        case 3:
            b.ascendingDisplay(b.getRoot());
            break;

        case 4:
            b.descendingDisplay(b.getRoot());
            break;

        case 5:
            cout<<"\nEnter word: ";
            cin>>word;
            cout<<"\nEnter meaning to be updated: ";
            cin>>meaning;
            b.updateNode(word, meaning);
            break;

        case 6:
            cout<<"\nEnter word to be deleted: ";
            cin>>word;
            b.deleteNode(word);
            break;
        case 0:
            cout<<"\n Ending the program";
            break;
```

```
                default:
                        cout<<"\nWrong choice";
                }


        }while(ch!=0);
        return 0;
}
```