

Experiment D19

```
#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);
    void inorder()
    {
        inorder_rec(root);
    }
    void postorder();

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

```

};
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;
}

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);
    }
}

void dictionary::create()
{
    int n;

```

```

string wordl,meaningl;
cout<<"\nHow many Word to insert?:\n";
cin>>n;
for(int i=0;i<n;i++)
{
    cout<<"\nEnter Word: ";
    cin>>wordl;
    cout<<"\nEnter Meaning: ";
    cin>>meaningl;
    insert(wordl,meaningl);
}
}
void dictionary::inorder_rec(node *rnode)
{
    if(rnode)
    {
        inorder_rec(rnode->left);
        cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
        inorder_rec(rnode->right);
    }
}
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 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;
    months.create();
    cout<<"Ascending 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";
    }
    return 0;
}

```