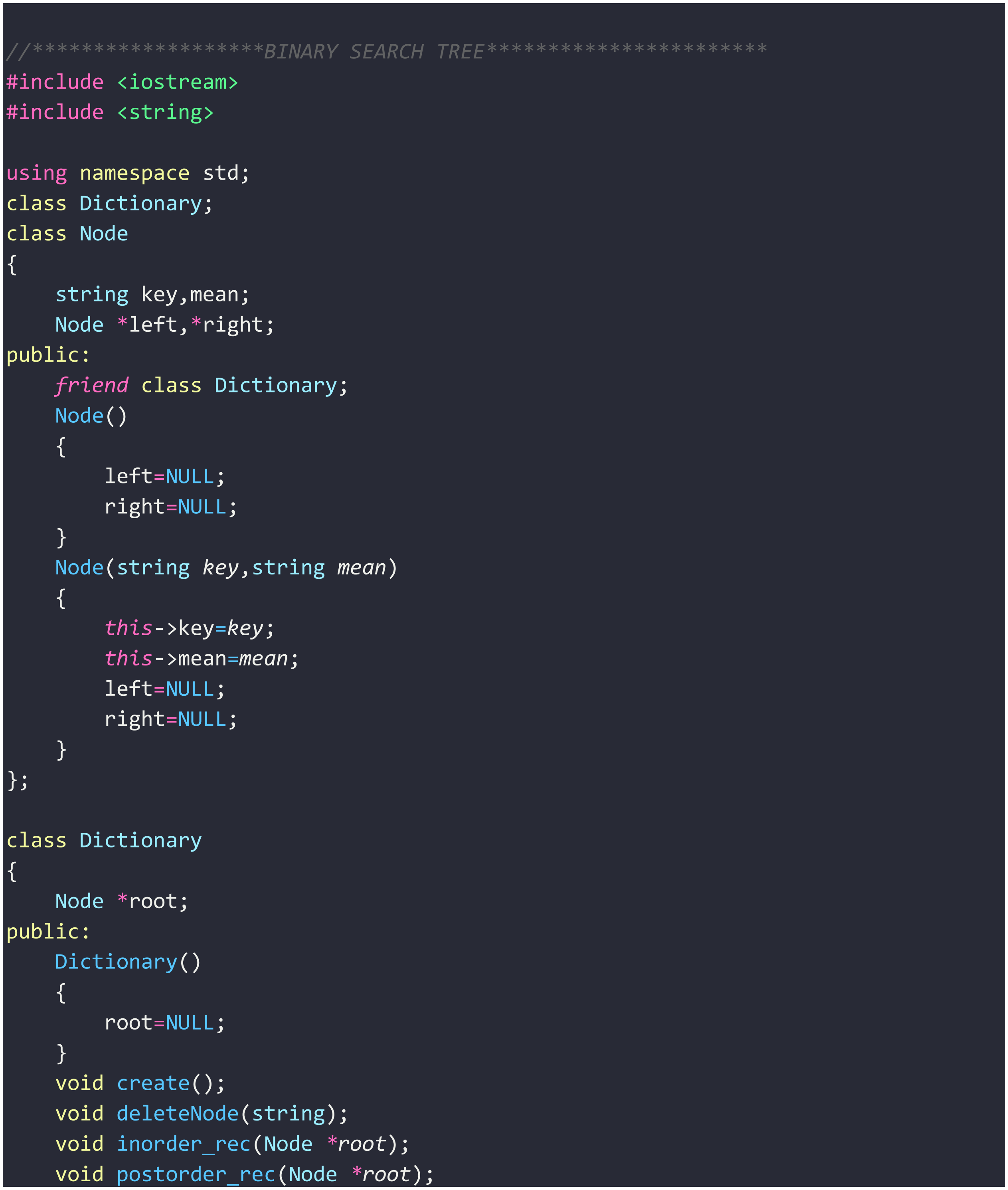
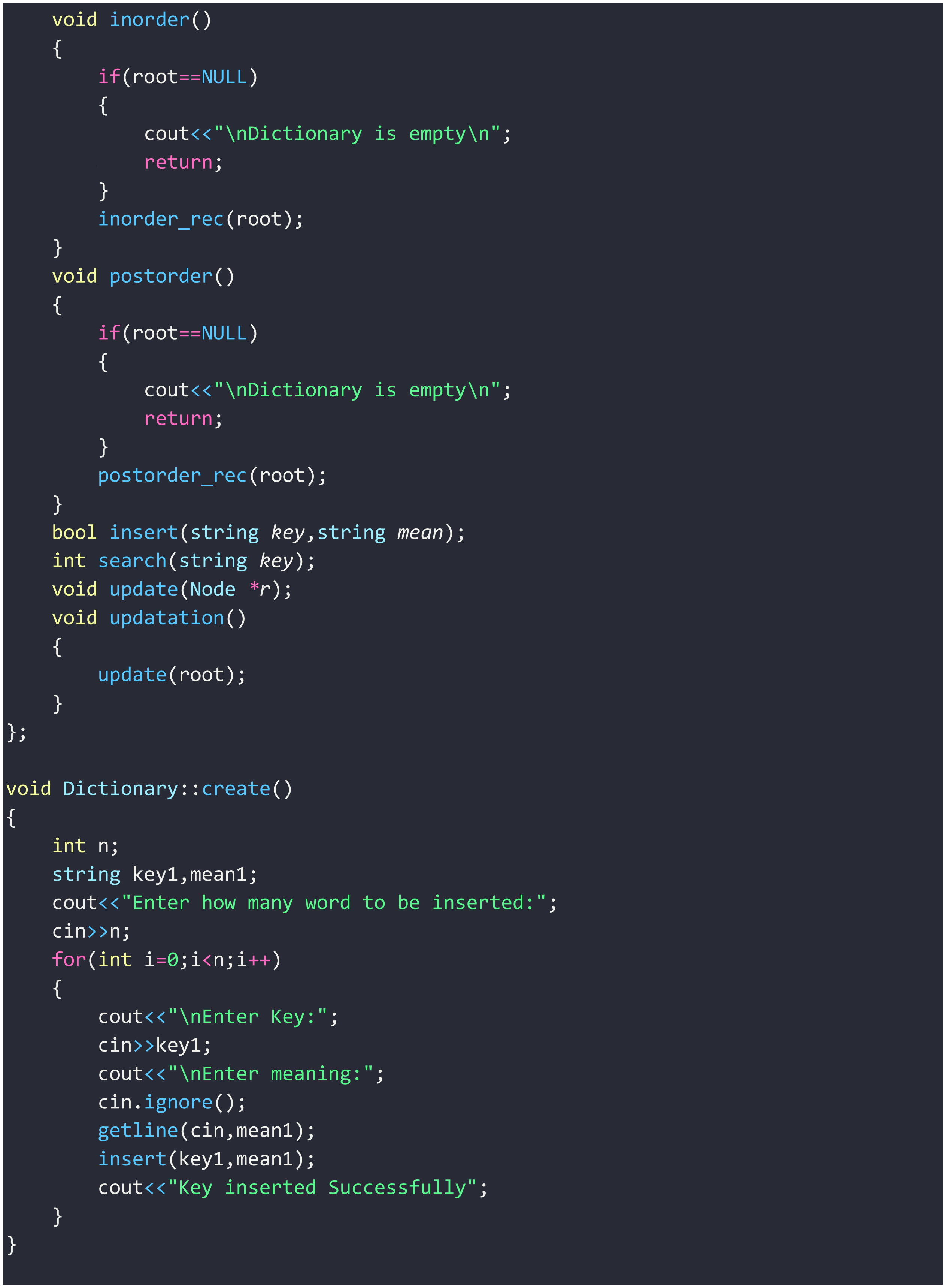
ASSIGNMENT NO: 1

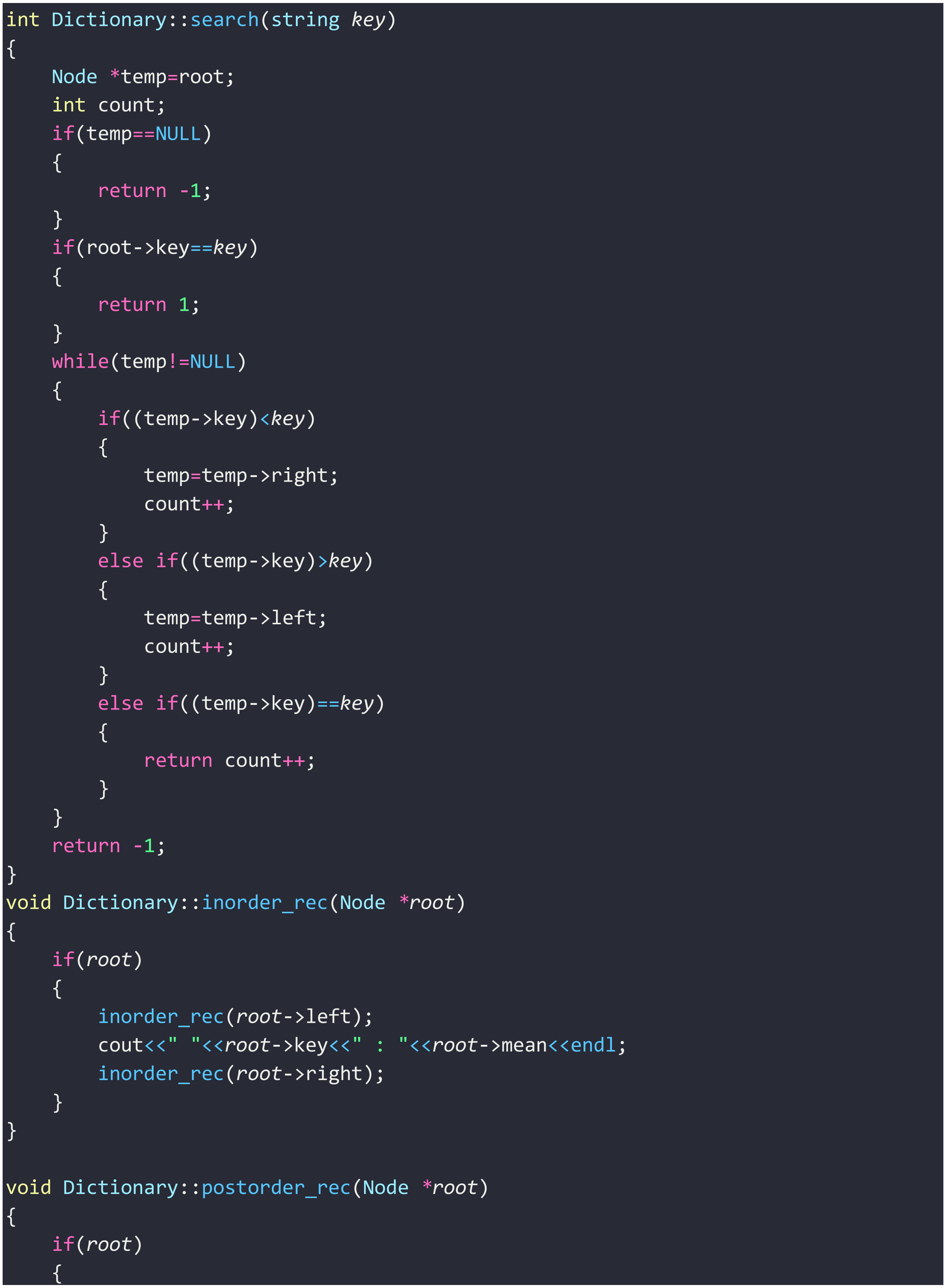
A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary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



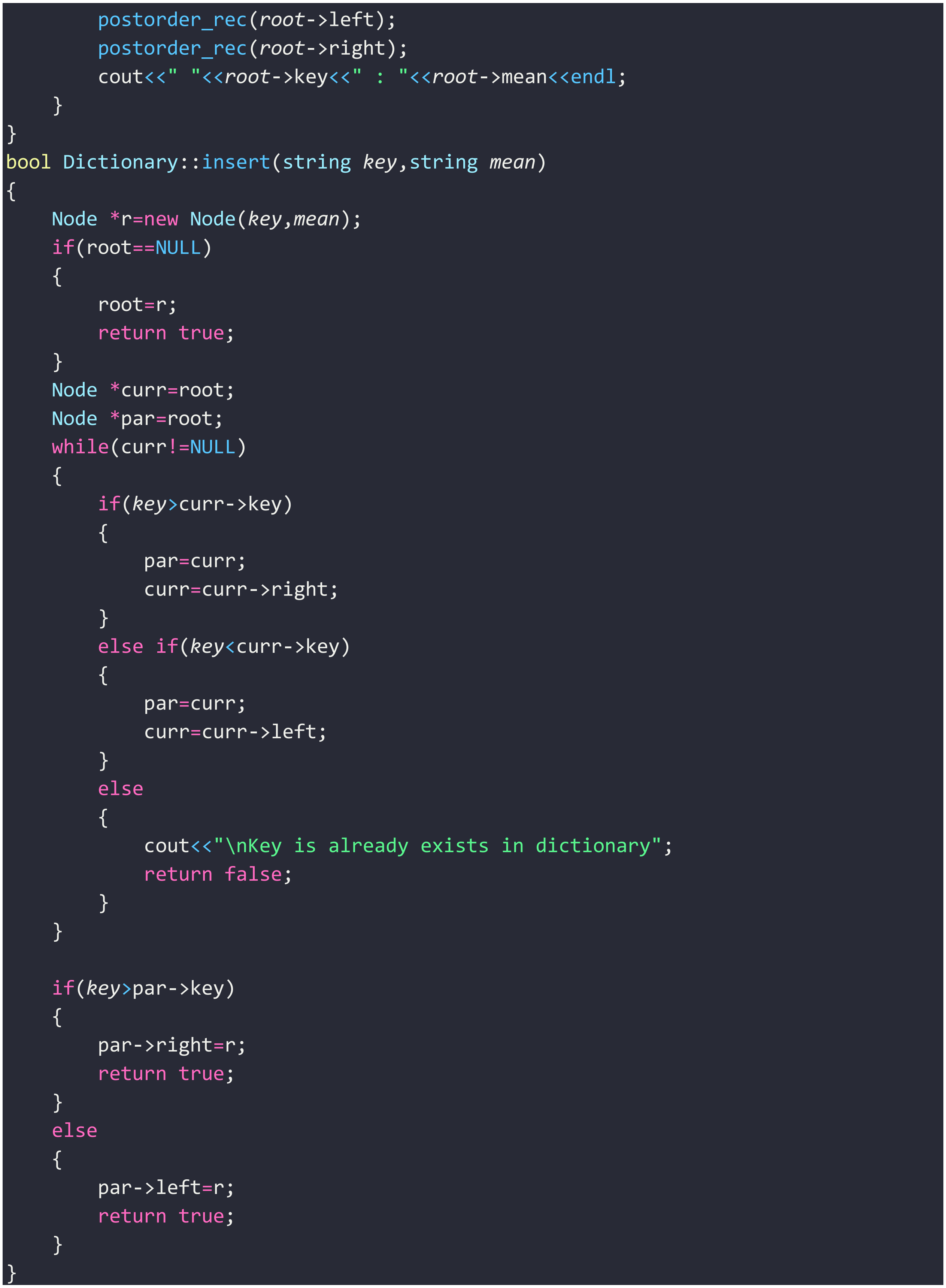
```
//*******************BINARY SEARCH TREE***********************
#include <iostream>
#include <string>

using namespace std;
class Dictionary;
class Node
{
    string key,mean;
    Node *left,*right;
public:
    friend class Dictionary;
    Node()
    {
        left=NULL;
        right=NULL;
    }
    Node(string key,string mean)
    {
        this->key=key;
        this->mean=mean;
        left=NULL;
        right=NULL;
    }
};

class Dictionary
{
    Node *root;
public:
    Dictionary()
    {
        root=NULL;
    }
    void create();
    void deleteNode(string);
    void inorder_rec(Node *root);
    void postorder_rec(Node *root);
```



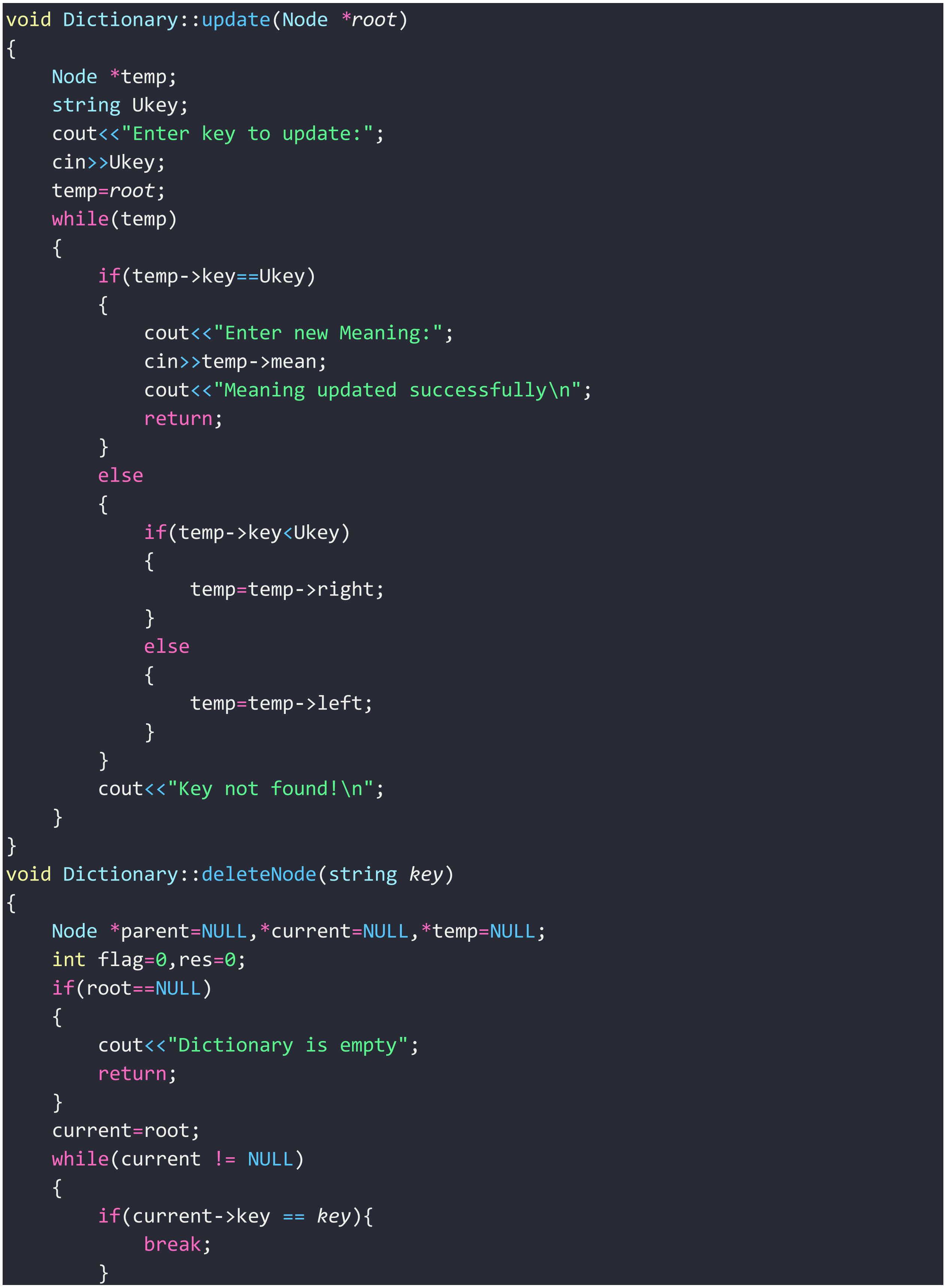
```
    void inorder()
    {
        if(root==NULL)
        {
            cout<<"\nDictionary is empty\n";
            return;
        }
        inorder_rec(root);
    }
    void postorder()
    {
        if(root==NULL)
        {
            cout<<"\nDictionary is empty\n";
            return;
        }
        postorder_rec(root);
    }
    bool insert(string key,string mean);
    int search(string key);
    void update(Node *r);
    void updatation()
    {
        update(root);
    }
};

void Dictionary::create()
{
    int n;
    string key1,mean1;
    cout<<"Enter how many word to be inserted:";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cout<<"\nEnter Key:";
        cin>>key1;
        cout<<"\nEnter meaning:";
        cin.ignore();
        getline(cin,mean1);
        insert(key1,mean1);
        cout<<"Key inserted Successfully";
    }
}
```



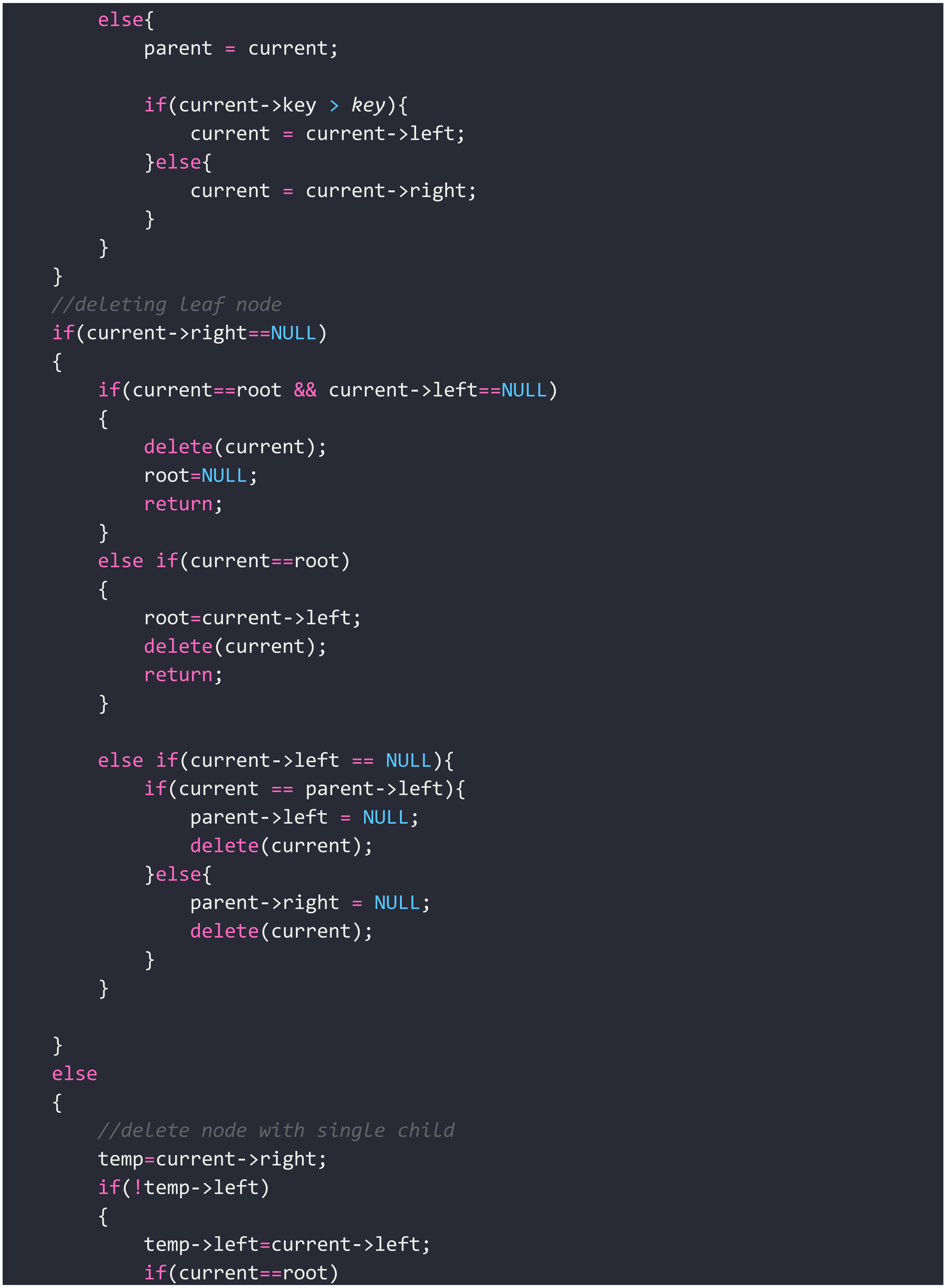
```
int Dictionary::search(string key)
{
    Node *temp=root;
    int count;
    if(temp==NULL)
    {
        return -1;
    }
    if(root->key==key)
    {
        return 1;
    }
    while(temp!=NULL)
    {
        if((temp->key)<key)
        {
            temp=temp->right;
            count++;
        }
        else if((temp->key)>key)
        {
            temp=temp->left;
            count++;
        }
        else if((temp->key)==key)
        {
            return count++;
        }
    }
    return -1;
}
void Dictionary::inorder_rec(Node *root)
{
    if(root)
    {
        inorder_rec(root->left);
        cout<<" "<<root->key<<" : "<<root->mean<<endl;
        inorder_rec(root->right);
    }
}

void Dictionary::postorder_rec(Node *root)
{
    if(root)
    {
```



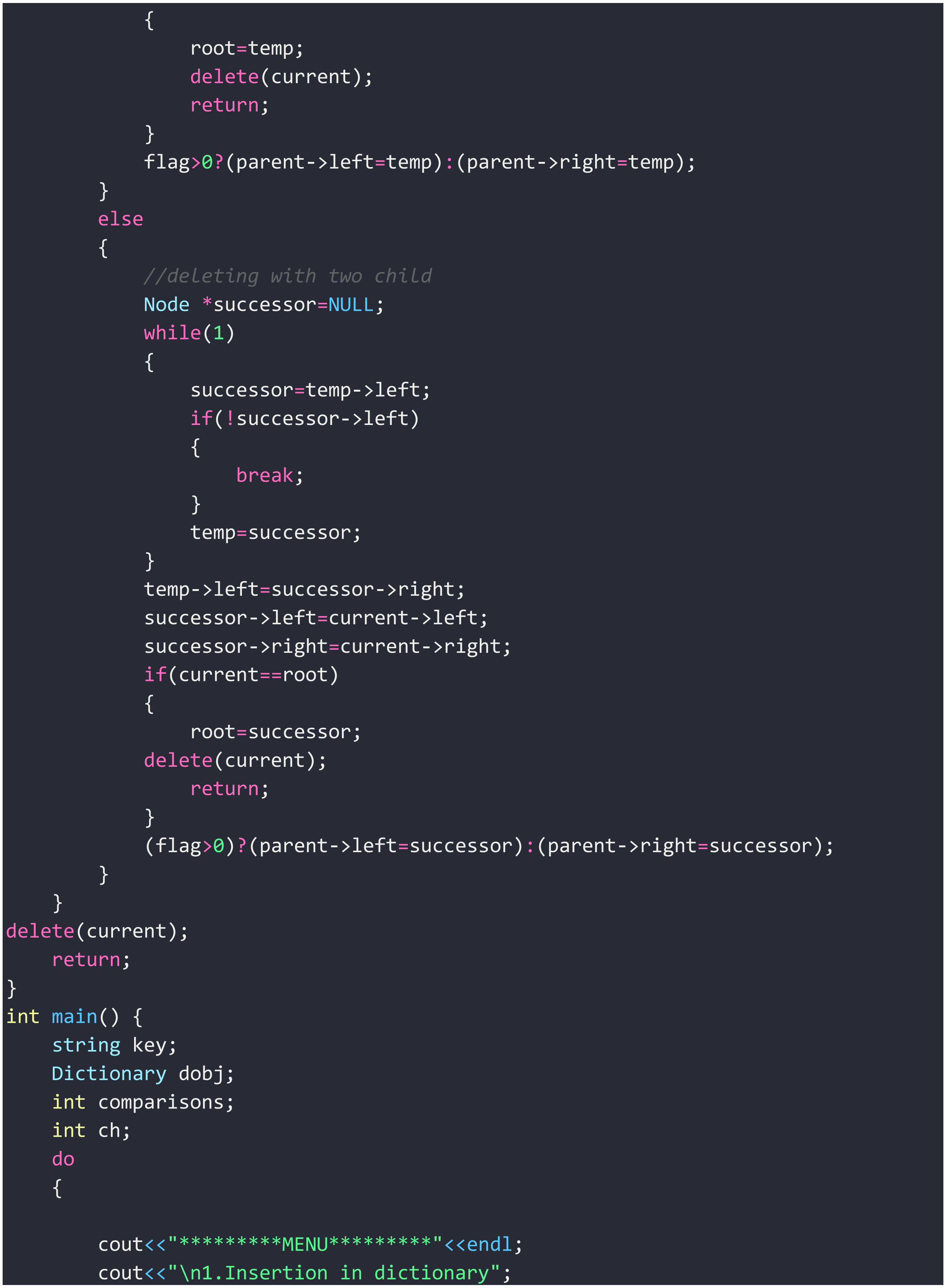
```
        postorder_rec(root->left);
        postorder_rec(root->right);
        cout<<" "<<root->key<<" : "<<root->mean<<endl;
    }
}
bool Dictionary::insert(string key,string mean)
{
    Node *r=new Node(key,mean);
    if(root==NULL)
    {
        root=r;
        return true;
    }
    Node *curr=root;
    Node *par=root;
    while(curr!=NULL)
    {
        if(key>curr->key)
        {
            par=curr;
            curr=curr->right;
        }
        else if(key<curr->key)
        {
            par=curr;
            curr=curr->left;
        }
        else
        {
            cout<<"\nKey is already exists in dictionary";
            return false;
        }
    }

    if(key>par->key)
    {
        par->right=r;
        return true;
    }
    else
    {
        par->left=r;
        return true;
    }
}
```



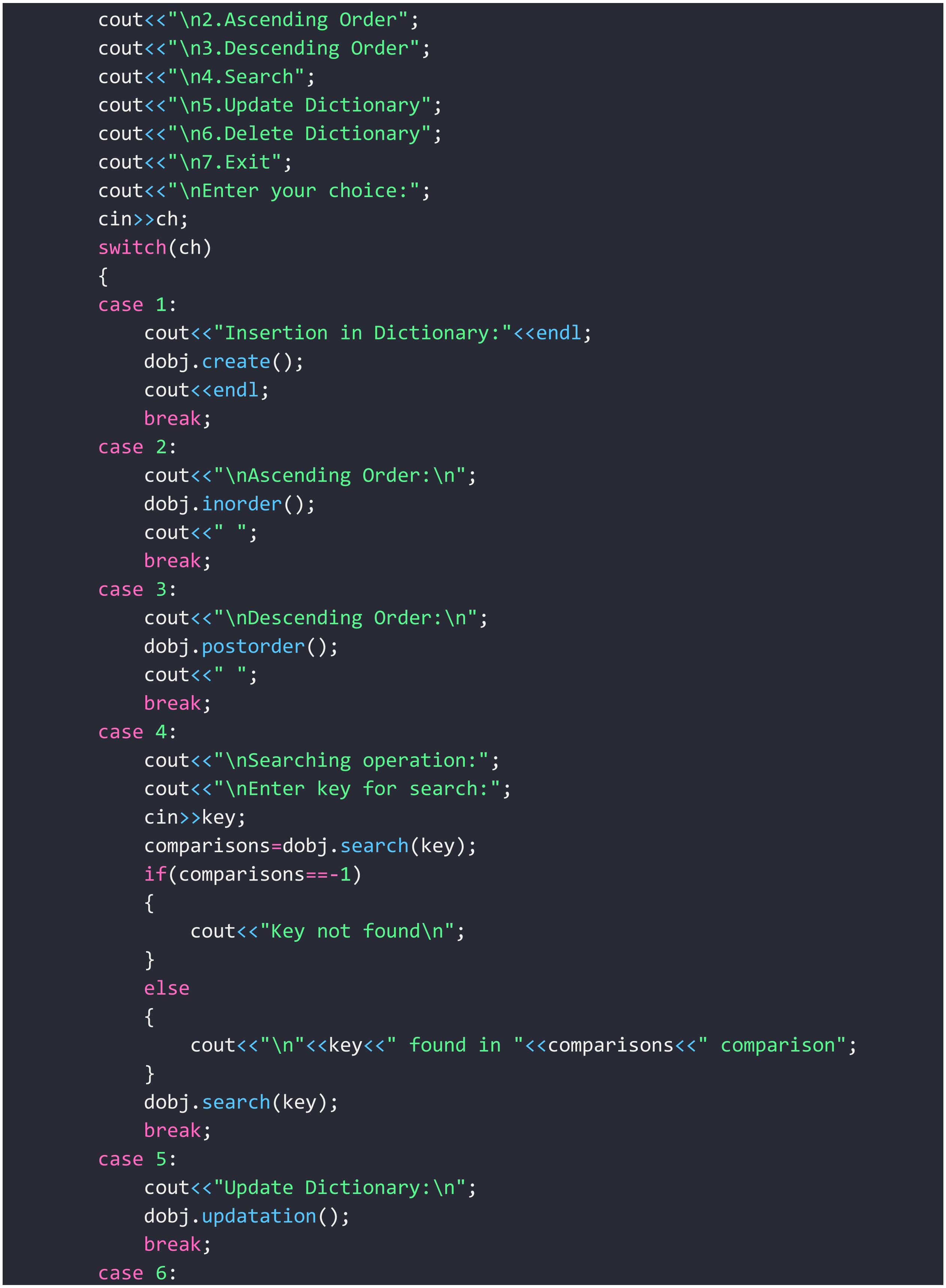
```
void Dictionary::update(Node *root)
{
    Node *temp;
    string Ukey;
    cout<<"Enter key to update:";
    cin>>Ukey;
    temp=root;
    while(temp)
    {
        if(temp->key==Ukey)
        {
            cout<<"Enter new Meaning:";
            cin>>temp->mean;
            cout<<"Meaning updated successfully\n";
            return;
        }
        else
        {
            if(temp->key<Ukey)
            {
                temp=temp->right;
            }
            else
            {
                temp=temp->left;
            }
        }
        cout<<"Key not found!\n";
    }
}
void Dictionary::deleteNode(string key)
{
    Node *parent=NULL,*current=NULL,*temp=NULL;
    int flag=0,res=0;
    if(root==NULL)
    {
        cout<<"Dictionary is empty";
        return;
    }
    current=root;
    while(current != NULL)
    {
        if(current->key == key){
            break;
        }
```



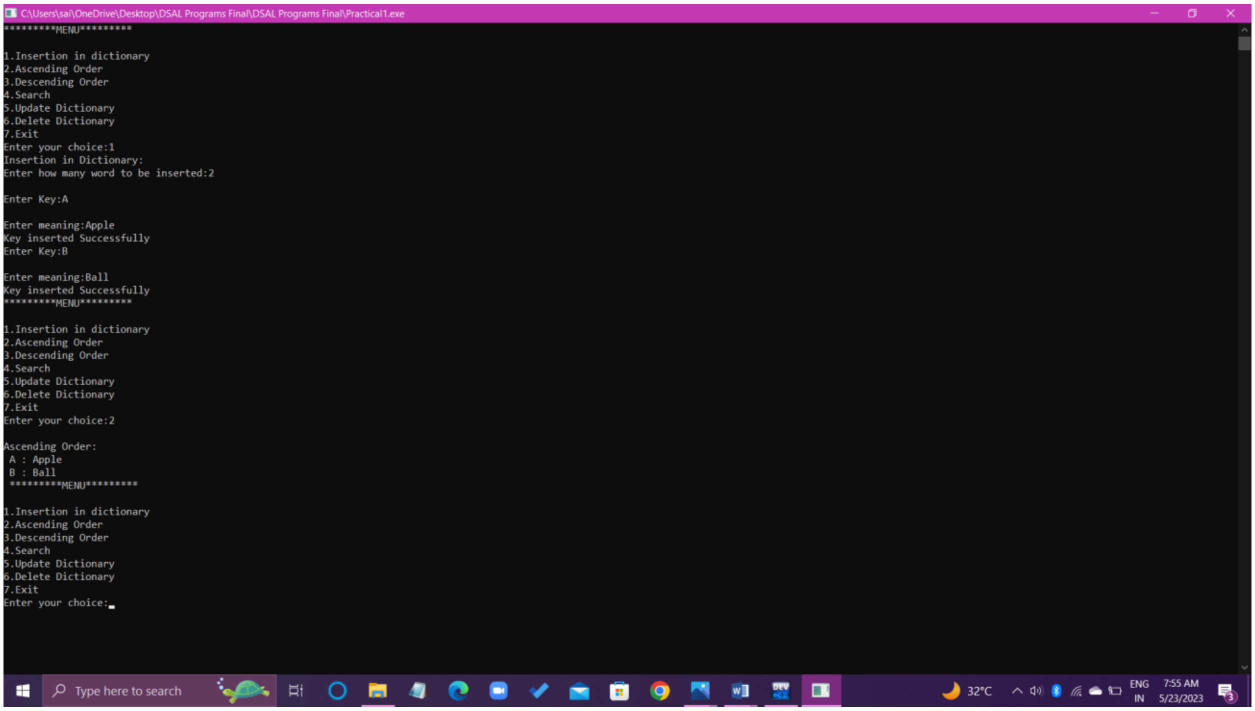
```
        else{
            parent = current;

            if(current->key > key){
                current = current->left;
            }else{
                current = current->right;
            }
        }
    }
    //deleting leaf node
    if(current->right==NULL)
    {
        if(current==root && current->left==NULL)
        {
            delete(current);
            root=NULL;
            return;
        }
        else if(current==root)
        {
            root=current->left;
            delete(current);
            return;
        }

        else if(current->left == NULL){
            if(current == parent->left){
                parent->left = NULL;
                delete(current);
            }else{
                parent->right = NULL;
                delete(current);
            }
        }

    }
    else
    {
        //delete node with single child
        temp=current->right;
        if(!temp->left)
        {
            temp->left=current->left;
            if(current==root)
```



```
            {
                root=temp;
                delete(current);
                return;
            }
            flag>0?(parent->left=temp):(parent->right=temp);
        }
        else
        {
            //deleting with two child
            Node *successor=NULL;
            while(1)
            {
                successor=temp->left;
                if(!successor->left)
                {
                    break;
                }
                temp=successor;
            }
            temp->left=successor->right;
            successor->left=current->left;
            successor->right=current->right;
            if(current==root)
            {
                root=successor;
            delete(current);
                return;
            }
            (flag>0)?(parent->left=successor):(parent->right=successor);
        }
    }
delete(current);
    return;
}
int main() {
    string key;
    Dictionary dobj;
    int comparisons;
    int ch;
    do
    {

        cout<<"*********MENU*********"<<endl;
        cout<<"\n1.Insertion in dictionary";
```



```
        cout<<"\n2.Ascending Order";
        cout<<"\n3.Descending Order";
        cout<<"\n4.Search";
        cout<<"\n5.Update Dictionary";
        cout<<"\n6.Delete Dictionary";
        cout<<"\n7.Exit";
        cout<<"\nEnter your choice:";
        cin>>ch;
        switch(ch)
        {
        case 1:
            cout<<"Insertion in Dictionary:"<<endl;
            dobj.create();
            cout<<endl;
            break;
        case 2:
            cout<<"\nAscending Order:\n";
            dobj.inorder();
            cout<<" ";
            break;
        case 3:
            cout<<"\nDescending Order:\n";
            dobj.postorder();
            cout<<" ";
            break;
        case 4:
            cout<<"\nSearching operation:";
            cout<<"\nEnter key for search:";
            cin>>key;
            comparisons=dobj.search(key);
            if(comparisons==-1)
            {
                cout<<"Key not found\n";
            }
            else
            {
                cout<<"\n"<<key<<" found in "<<comparisons<<" comparison";
            }
            dobj.search(key);
            break;
        case 5:
            cout<<"Update Dictionary:\n";
            dobj.updatation();
            break;
        case 6:
            cout<<"Deleting Node\n";
            cout<<"\nEnter key to delete:";
            cin>>key;
            dobj.deleteNode(key);
            break;
        case 7:
            cout<<"Thank you for using this program";
            exit(0);
            break;
        }
    }while(ch!=7);

    return 0;
}
```


C:\Users\sai\OneDrive\Desktop\DSAL Programs Final\DSAL Programs Final\Practical1.exe
Type here to search
32°C
ENG
IN
7:55 AM
5/23/2023

```
*********MENU*********

1.Insertion in dictionary
2.Ascending Order
3.Descending Order
4.Search
5.Update Dictionary
6.Delete Dictionary
7.Exit
Enter your choice:1
Insertion in Dictionary:
Enter how many word to be inserted:2

Enter Key:A

Enter meaning:Apple
Key inserted Successfully
Enter Key:B

Enter meaning:Ball
Key inserted Successfully
*********MENU*********

1.Insertion in dictionary
2.Ascending Order
3.Descending Order
4.Search
5.Update Dictionary
6.Delete Dictionary
7.Exit
Enter your choice:2

Ascending Order:
 A : Apple
 B : Ball
 *********MENU*********

1.Insertion in dictionary
2.Ascending Order
3.Descending Order
4.Search
5.Update Dictionary
6.Delete Dictionary
7.Exit
Enter your choice:
```