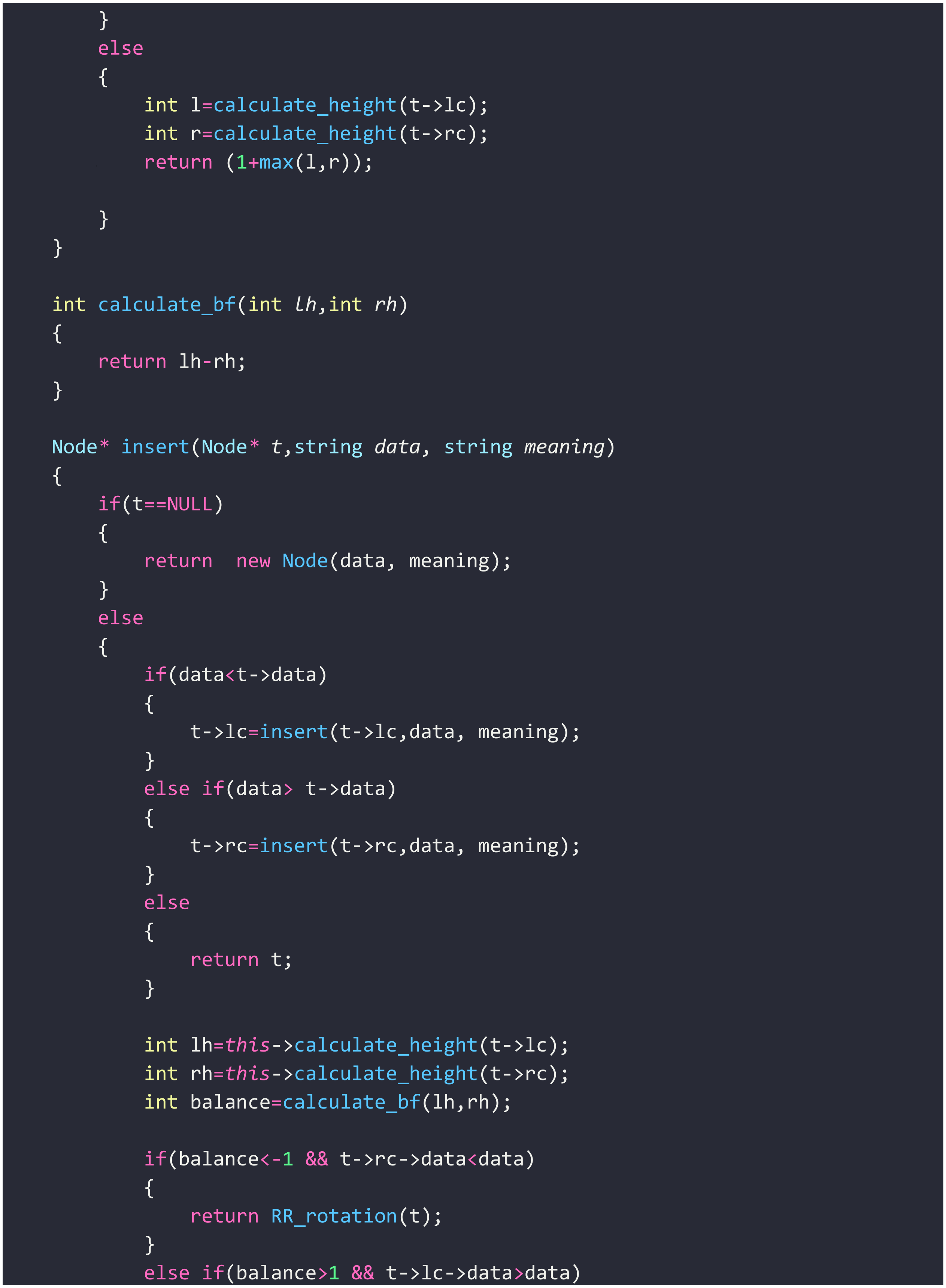
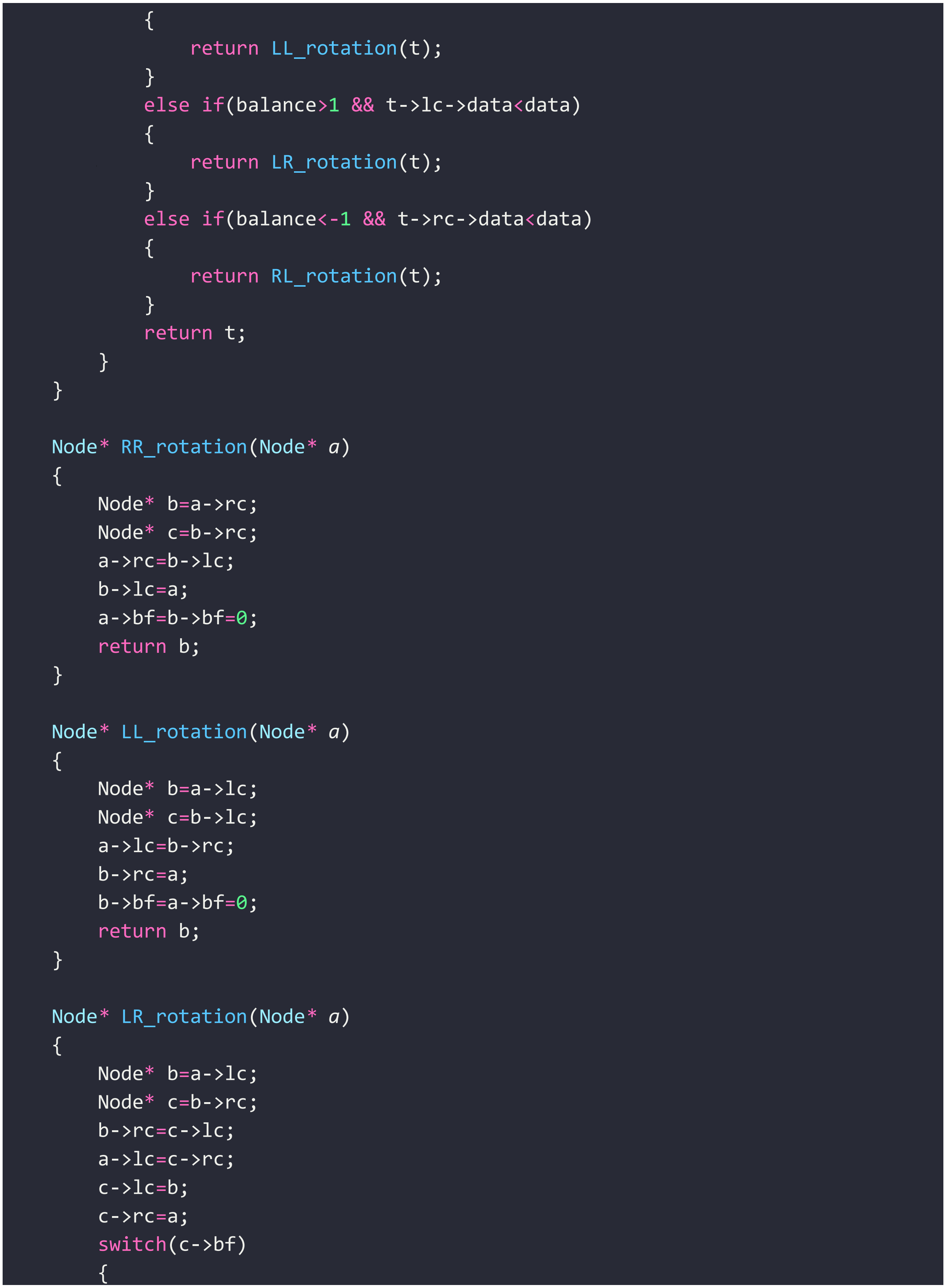
ASSIGNMENT NO – 9

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 Height balance tree and find the complexity for finding a keyword.

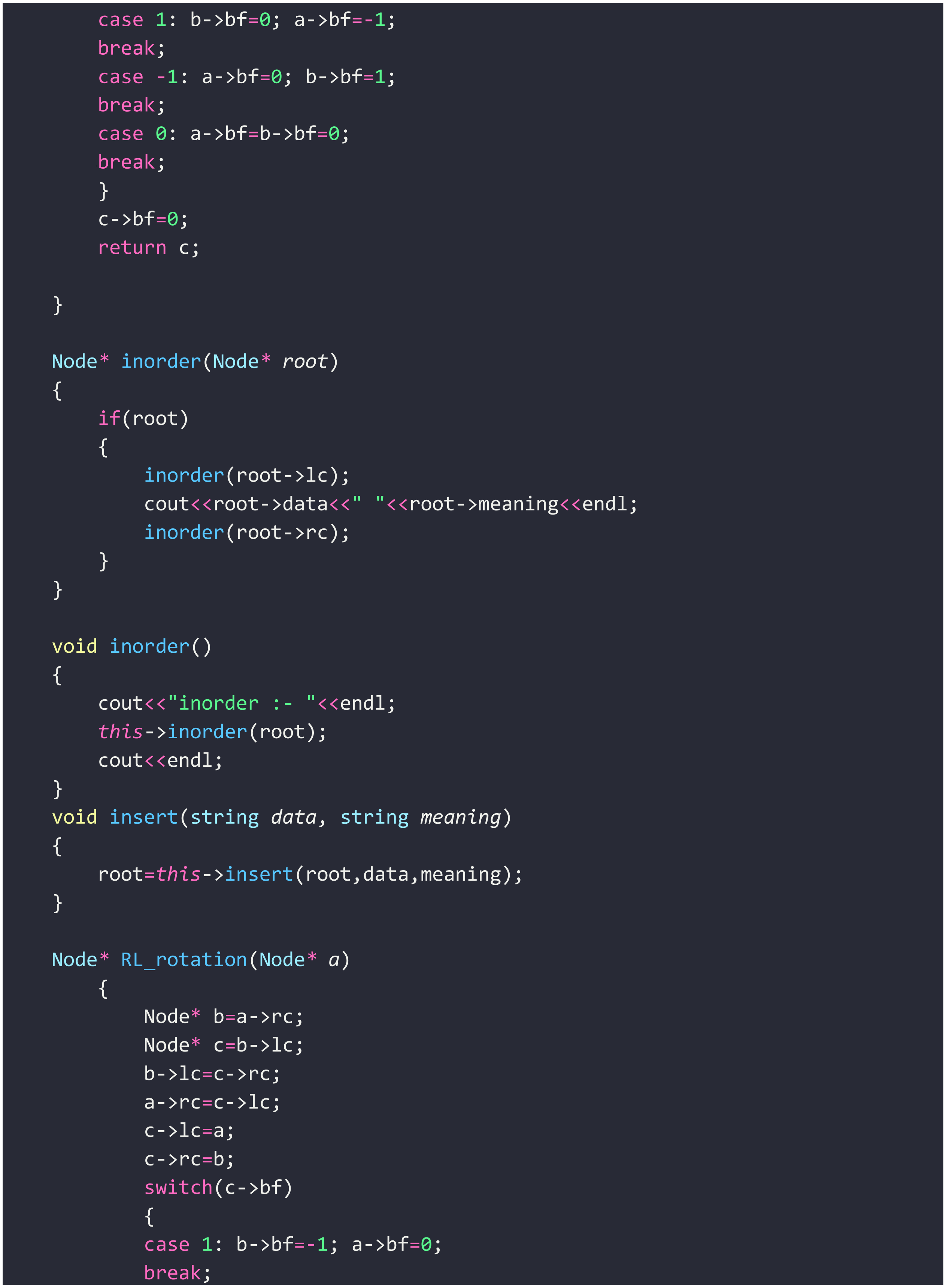
```
#include <iostream>
using namespace std;
class Node{
    string data;
    string meaning;
    int bf;
    Node* lc;
    Node* rc;
public:
    Node()
    {
        data="";
        meaning="";
        lc=NULL;
        rc=NULL;
        bf=0;
    }
    Node(string data, string meaning)
    {
        this->data=data;
        this->meaning = meaning;
        lc=NULL;
        rc=NULL;
        bf=0;
    }
    friend class AVL;
};
class AVL{
    Node* root;
public:
    AVL()
    {
        root=NULL;
    }
    int calculate_height(Node* t)
    {
        if(t==NULL)
        {
            return -1;
```



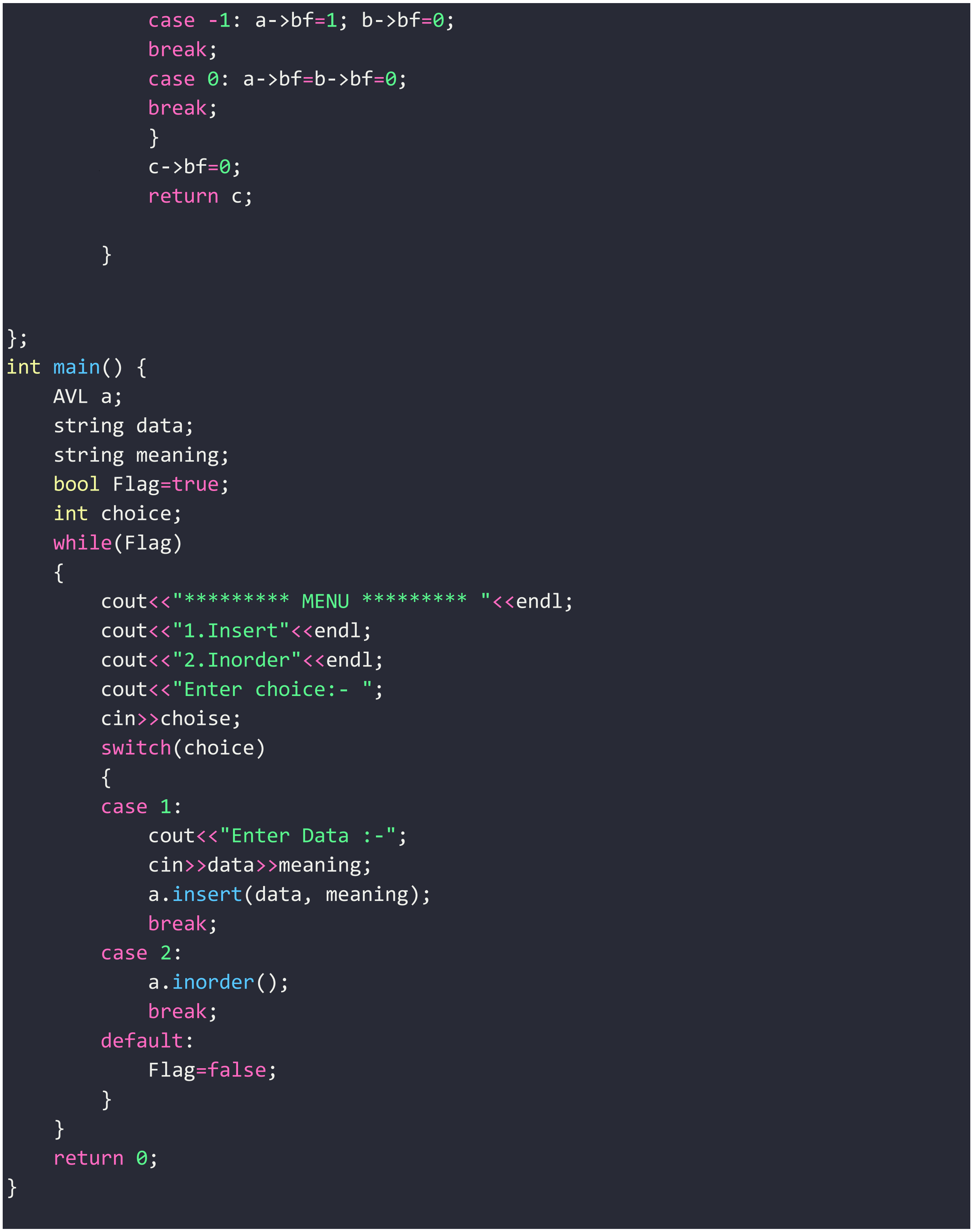
```
        }
        else
        {
            int l=calculate_height(t->lc);
            int r=calculate_height(t->rc);
            return (1+max(l,r));

        }
    }

    int calculate_bf(int lh,int rh)
    {
        return lh-rh;
    }

    Node* insert(Node* t,string data, string meaning)
    {
        if(t==NULL)
        {
            return  new Node(data, meaning);
        }
        else
        {
            if(data<t->data)
            {
                t->lc=insert(t->lc,data, meaning);
            }
            else if(data> t->data)
            {
                t->rc=insert(t->rc,data, meaning);
            }
            else
            {
                return t;
            }

            int lh=this->calculate_height(t->lc);
            int rh=this->calculate_height(t->rc);
            int balance=calculate_bf(lh,rh);

            if(balance<-1 && t->rc->data<data)
            {
                return RR_rotation(t);
            }
            else if(balance>1 && t->lc->data>data)
```



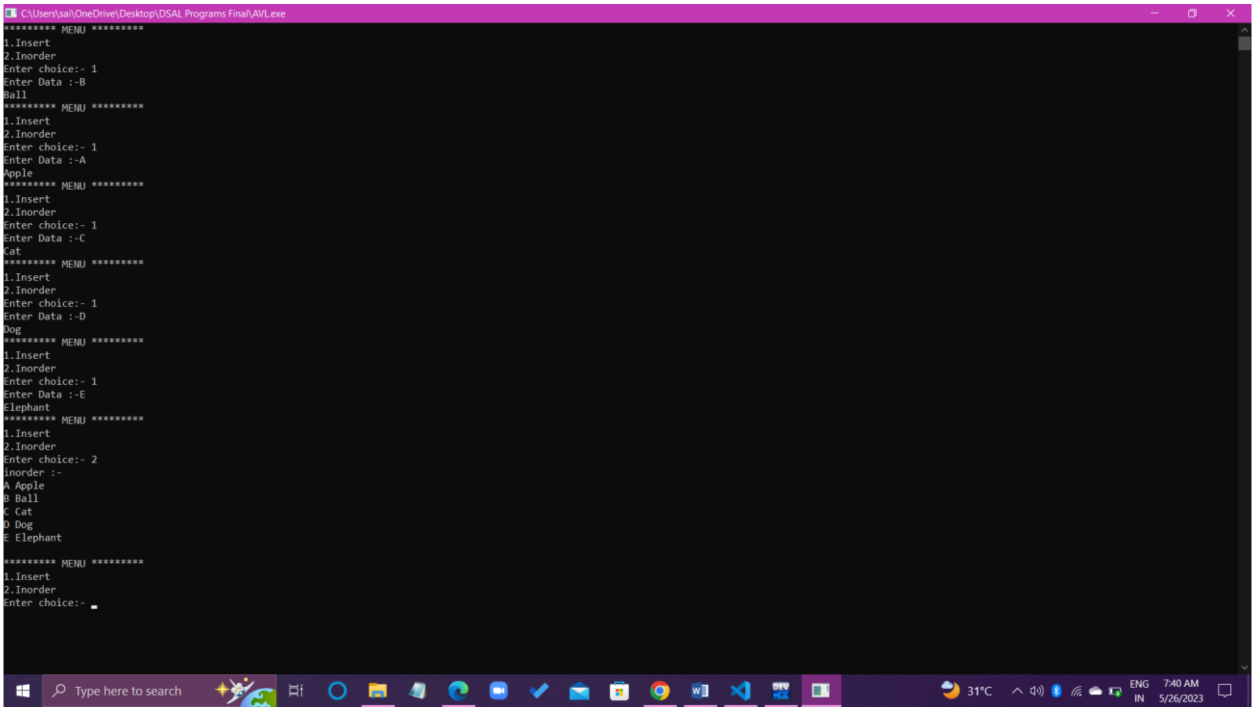
```
        {
            return LL_rotation(t);
        }
        else if(balance>1 && t->lc->data<data)
        {
            return LR_rotation(t);
        }
        else if(balance<-1 && t->rc->data<data)
        {
            return RL_rotation(t);
        }
        return t;
    }
}

Node* RR_rotation(Node* a)
{
    Node* b=a->rc;
    Node* c=b->rc;
    a->rc=b->lc;
    b->lc=a;
    a->bf=b->bf=0;
    return b;
}

Node* LL_rotation(Node* a)
{
    Node* b=a->lc;
    Node* c=b->lc;
    a->lc=b->rc;
    b->rc=a;
    b->bf=a->bf=0;
    return b;
}

Node* LR_rotation(Node* a)
{
    Node* b=a->lc;
    Node* c=b->rc;
    b->rc=c->lc;
    a->lc=c->rc;
    c->lc=b;
    c->rc=a;
    switch(c->bf)
    {
```



```
        case 1: b->bf=0; a->bf=-1;
        break;
        case -1: a->bf=0; b->bf=1;
        break;
        case 0: a->bf=b->bf=0;
        break;
        }
        c->bf=0;
        return c;

    }

    Node* inorder(Node* root)
    {
        if(root)
        {
            inorder(root->lc);
            cout<<root->data<<" "<<root->meaning<<endl;
            inorder(root->rc);
        }
    }

    void inorder()
    {
        cout<<"inorder :- "<<endl;
        this->inorder(root);
        cout<<endl;
    }
    void insert(string data, string meaning)
    {
        root=this->insert(root,data,meaning);
    }

    Node* RL_rotation(Node* a)
        {
            Node* b=a->rc;
            Node* c=b->lc;
            b->lc=c->rc;
            a->rc=c->lc;
            c->lc=a;
            c->rc=b;
            switch(c->bf)
            {
            case 1: b->bf=-1; a->bf=0;
            break;
```



```
            case -1: a->bf=1; b->bf=0;
            break;
            case 0: a->bf=b->bf=0;
            break;
            }
            c->bf=0;
            return c;

        }


};
int main() {
    AVL a;
    string data;
    string meaning;
    bool Flag=true;
    int choice;
    while(Flag)
    {
        cout<<"********* MENU ********* "<<endl;
        cout<<"1.Insert"<<endl;
        cout<<"2.Inorder"<<endl;
        cout<<"Enter choice:- ";
        cin>>choise;
        switch(choice)
        {
        case 1:
            cout<<"Enter Data :-";
            cin>>data>>meaning;
            a.insert(data, meaning);
            break;
        case 2:
            a.inorder();
            break;
        default:
            Flag=false;
        }
    }
    return 0;
}
```



```
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 1
Enter Data :-B
Ball
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 1
Enter Data :-A
Apple
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 1
Enter Data :-C
Cat
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 1
Enter Data :-D
Dog
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 1
Enter Data :-E
Elephant
********* MENU *********
1.Insert
2.Inorder
Enter choice:- 2
inorder :-
A Apple
B Ball
C Cat
D Dog
E Elephant

********* MENU *********
1.Insert
2.Inorder
Enter choice:-
```