#include <iostream>

#include <vector>

#include<algorithm>

#include<conio.h>

using namespace std;

class Node

{

public:

    string label;

    vector<Node\*> children;

    Node(const string label)

    {

        this->label = label;

    }

    void addChild(Node\* child)

    {

        children.push\_back(child);

    }

    void printTree(int depth = 0)

    {

        for (int i = 0; i < depth; ++i)

        {

            cout << "-";

        }

        cout << label <<endl ;

        for (auto child : children)

        {

            child->printTree(depth + 1);

        }

    }

};

class GrammaticalTree

{

public:

    vector<string> splitSentence(string sentence)

    {

        vector<string> words;

        string word = "";

        for (char c : sentence)

        {

            if (c == ' ')

            {

                words.push\_back(word);

                word = "";

            }

            else

            {

                word += c;

            }

        }

        words.push\_back(word);

        return words;

    }

    Node\* createSubject(vector<string> words)

    {

        Node\* subject = new Node("Subject (NP)");

        int count = 0;

        for (int i = 0; i < words.size(); ++i)

        {

            if (count == 4)

            {

                break;

            }

            Node\* node = new Node(words[i]);

            subject->addChild(node);

            count++;

        }

        return subject;

    }

    Node\* createNoun(vector<string> words)

    {

        Node\* noun = new Node("Noun");

        noun->addChild(new Node(words[3]));

        noun->addChild(new Node(words[8]));

        return noun;

    }

    Node\* createVerb(vector<string> words)

    {

        Node\* verb = new Node("Verb (V)");

        verb->addChild(new Node(words[4]));

        return verb;

    }

    Node\* createPreposition(vector<string> words)

    {

        Node\* preposition = new Node("preposition");

        preposition->addChild(new Node(words[5]));

        return preposition;

    }

    Node\* CreateAdjective(vector<string> words)

    {

        Node\* Adjective = new Node("adjective");

        Adjective->addChild(new Node(words[1]));

        Adjective->addChild(new Node(words[2]));

        Adjective->addChild(new Node(words[7]));

        return Adjective;

    }

    Node\* createDeterminer(vector<string> words)

    {

        Node\* determiner = new Node("determiner");

        determiner->addChild(new Node(words[0]));

        determiner->addChild(new Node(words[6]));

        return determiner;

    }

    Node\* createObject(vector<string> words)

    {

        Node\* object = new Node("Object (NP)");

        for (int i = 5; i < words.size(); i++)

        {

            Node\* node = new Node(words[i]);

            object->addChild(node);

        }

        return object;

    }

    Node\* constructTree(string sentence)

    {

        Node\* root = new Node("Sentence (S");

        vector<string> words = splitSentence(sentence);

        root->addChild(createSubject(words));

        root->addChild(createVerb(words));

        root->addChild(createObject(words));

        root->addChild(createDeterminer(words));

        root->addChild(CreateAdjective(words));

        root->addChild(createNoun(words));

        root->addChild(createPreposition(words));

        return root;

    }

};

void title()

{

    cout << "\t\t=======================\n";

    cout << "\t\t    GRAMMATICAL TREE \n";

    cout << "\t\t=======================\n";

}

int main()

{

    lable:

    system("cls");

    title();

    vector<string> wordsList = {"The","black", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog", "near", "park", "sleeps", "swiftly", "chair", "tree", "car", "computer", "happy", "sun", "moon"};

    cout << endl << endl;

    string userInput;

    cout << "Enter a sentence: ";

    getline(cin, userInput);

    bool isValid = true;

    vector<string> userWords = GrammaticalTree().splitSentence(userInput);

    for (auto userWord : userWords)

    {

        if (find(wordsList.begin(), wordsList.end(), userWord) == wordsList.end())

        {

            isValid = false;

            break;

        }

    }

    if (isValid)

    {

        GrammaticalTree tree;

        Node\* root = tree.constructTree(userInput);

        root->printTree();

    } else {

        cout << "Invalid input. Please enter the correct sentence.\n" << endl;

        cout<<"Press enter to continue";

        getch();

        goto lable;

    }

}



