МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт информационных технологий

Кафедра «Информационные технологии и компьютерные системы»

ОТЧЕТ

по лабораторной работе №4

по дисциплине “Системное программное обеспечение”

Вариант 4

Выполнил:
ст. гр. ИТ/б-22-6-о Донец Н.О.
Принял:
ассистент Ткаченко К.С.

Севастополь
2024 г.

Цель работы:

Изучить метод рекурсивного спуска, а также способы построения синтаксических анализаторов.

Задание:

Разработать и отладить программу синтаксического анализатора методом рекурсивного спуска, которая должна быть оформлена в виде отдельной процедуры (подпрограммы).

Грамматика языка Logic4:

<программа>::=<блок>

<блок>::=<оператор>|<оператор>;< блок >

<оператор>:=<переменная>:=<выражение>

<оператор>:= if <переменная> ? <оператор> : <оператор>

<выражение>::=<фактор>|<выражение>#<фактор>

<фактор>::=<первичное>|<фактор>&<первичное>

<первичное>::=<идент.>|<константа>|(<выражение>)

<константа>::=<целая константа>

<целая константа>::=<число>

<число>::=<цифра>|<число><цифра>

<цифра>::=0|1|2|3|4|5|6|7|8|9

<идент.>::=<буква>|<идент.><буква>

<буква>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

Ход работы:

Для того чтобы можно было воспользоваться методом рекурсивного спуска необходимо выполнить некоторые преобразования заданной по варианту грамматики, а именно заменить вывод леворекурсивных нетерминалов на праворекурсивные.

<выражение>::=<фактор>|<выражение>#<фактор>
Обозначим выражение E, фактор F, тогда исходное правило имеет вид:
E → F | E#F
Далее преобразуем это правило так, чтобы избавиться от левой рекурсии.
E → FE’
E’ → #FE’ | ε

Те же манипуляции проводим с фактором.
<фактор>::=<первичное>|<фактор>&<первичное>
Обозначим первичное за P, тогда исходное правило имеет вид:
F → P | F&P
Далее преобразуем это правило так, чтобы избавиться от левой рекурсии.
F → PF’
F’ → &PF’ | ε

Был разработан класс синтаксического анализатора, решающий поставленную задачу (Листинг 1).

Листинг 1 – Синтаксический анализатор

```
#ifndef SYNTAXIS_ANALYZER
#define SYNTAXIS_ANALYZER

#include <iostream>
#include <map>
#include <vector>
#include <vector>
#include <sstream>
```

```
#include <algorithm>

class SyntaxisAnalyzer {
    private:
        int head = 0;

        int validateOperatorIf(std::string);
        int validateFactor(std::string);
        int validateOperator(std::string);
        int validateOperatorAR(std::string);
        int validateExpression(std::string);
        int validateBlock(std::string);
        int validatePrimary(std::string);
    public:
        SyntaxisAnalyzer() {}
        bool analyze(std::string str);
};

#endif

#include "SyntaxisAnalyzer.h"

bool SyntaxisAnalyzer::analyze(std::string str) {
    head = 0;
    if (validateBlock(str)) {
        return false;
    }
    return true;
}

int SyntaxisAnalyzer::validateBlock(std::string str) {
    std::cout<< "Block" << std::endl;
    if(validateOperator(str)) {
        std::cout<< "No block " << std::endl;
        return -1;
    }
    if (str[head] == ';') {
        head++;
        if(validateBlock(str)) return -1;
    }
    else if (head == str.size()) {
        return 0;
    }
    else {
        return -1;
    }
    return 0;
}

int SyntaxisAnalyzer::validateOperator(std::string str) {
    std::cout<< "Operator " << "head -> " << str[head] << std::endl;
    if (str[head] == 'I') {
        head++;
        if(validateOperatorAR(str)) return -1;
    }
    else if (str[head] == 'T') {
        head++;
        if(validateOperatorIf(str)) return -1;
    }
    else {
        return -1;
    }
    return 0;
```

```
}

int SyntaxisAnalyzer::validateOperatorAR(std::string str) {
    std::cout<< "Operator AR " << "head -> " << str[head] << std::endl;
    if (str[head] == 'A' || str[head] == 'R') {
        head++;
        if (validateExpression(str)) {
            return -1;
        }
    }
    else {
        std::cout<< "No operator AR " << std::endl;
        return -1;
    }
    return 0;
}

int SyntaxisAnalyzer::validateOperatorIf(std::string str) {
    std::cout<< "Operator IF " << "head -> " << str[head] << std::endl;
    if (str[head] == 'I') {
        head++;
    }
    else {
        std::cout<< "No Term 1 " << std::endl;
        return -1;
    }

    if (str[head] == '?') {
        head++;
        if (validateOperator(str)) return -1;
    }
    else {
        std::cout<< "No Term 2 " << std::endl;
        return -1;
    }

    if (str[head] == ':') {
        head++;
        if (validateOperator(str)) return -1;
    }
    else {
        std::cout<< "No Term 3 " << std::endl;
        return -1;
    }
    return 0;
}

int SyntaxisAnalyzer::validateExpression(std::string str) {
    std::cout<< "Expression " << "head -> " << str[head] << std::endl;
    if (validateFactor(str)) {
        std::cout<< "No expression " << std::endl;
        return -1;
    }
    if (str[head] == 'E') {
        head++;
        if (validateExpression(str)) return -1;
    }
    return 0;
}

int SyntaxisAnalyzer::validateFactor(std::string str) {
    std::cout<< "Factor " << "head -> " << str[head] << std::endl;
    if (validatePrimary(str)) {
```

```
        std::cout<< "No factor " << std::endl;
        return -1;
    }
    if (str[head] == 'F') {
        head++;
        if (validateFactor(str)) return -1;
    }
    return 0;
}

int SyntaxisAnalyzer::validatePrimary(std::string str) {
    std::cout<< "Primary " << "head -> " << str[head] << std::endl;
    if (str[head] == 'I' || str[head] == 'C') {
        head++;
        return 0;
    }
    else if (str[head] == '(') {
        head++;
        if (validateExpression(str)) return -1;
        if (str[head] == ')') {
            head++;
            return 0;
        }
    }
    else {
        std::cout<< "No primary " << std::endl;
        return -1;
    }
    return 0;
}


}
```

Также были проведены тесты работы анализатора (рисунки 1 – 2).

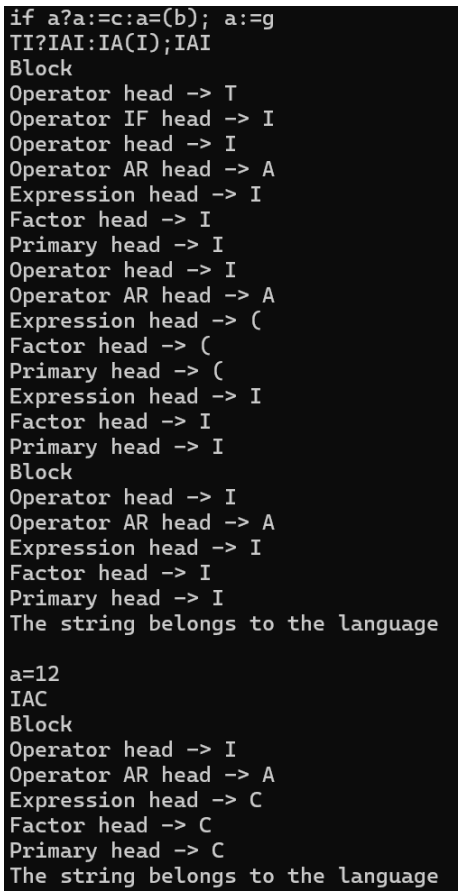

Рисунок 1 – Первый тест

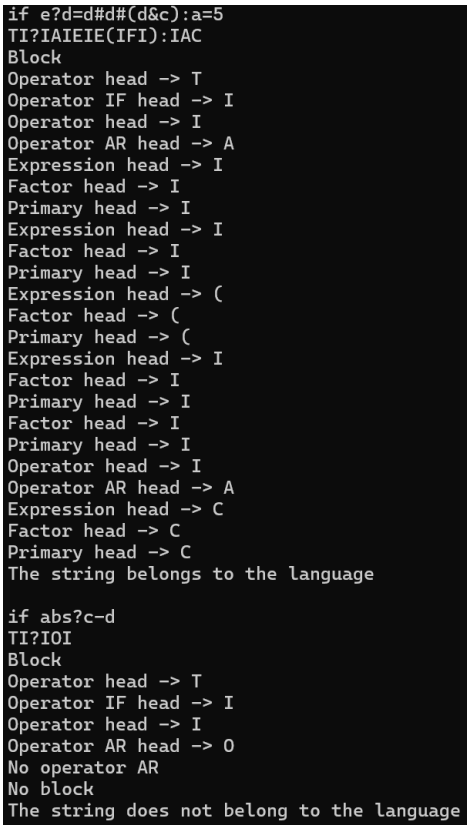

Рисунок 2 – Второй тест

Выводы

В ходе лабораторной работы был изучен метод рекурсивного спуска, а также были изучены способы построения синтаксических анализаторов.