Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет им. Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА: **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ: 09.03.01 **Информатика и вычислительная техника**

О Т Ч Е Т

по домашнему заданию 2

Тема: ЛЕКСИЧЕСКИЕ И СИНТАКСИЧЕСКИЕ АНАЛИЗАТОРЫ

Дисциплина: МЗЯиОК

Студент	ИУ6-43Б	14.05.2024	В.К. Залыгин
	(группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Я.С. Петрова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы

Целью домашнего задания № 2 по дисциплине «Машинно-зависимые языки и основы компиляции» является закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.

Задание

Разработать программу, которая выполняет лексический и синтаксический анализ указанных конструкций языков Паскаль или C++. Программа должна обеспечивать многократный ввод предложений, их обработку с выводом на экран результатов лексического и синтаксического анализов и завершать работу при вводе слова "все". Для каждого введенного предложения анализатор должен возвращать «Конструкция распознана» или «Обнаружена ошибка».

При выполнении задания студент должен:

- разработать, записать в форме Бекуса-Наура и изобразить в виде синтаксических диаграмм грамматику заданных конструкций формального языка;

- используя формальные признаки определить тип грамматики по классификации Хомского;

- проанализировать правила грамматики и выбрать метод синтаксического анализа конструкций языка;

- в соответствии с выбранным методом синтаксического анализа выбрать способ реализации лексического анализа: построение подпрограммы сканера или использование распознавателей лексем по мере разбора предложений языка;

- при необходимости выбрать и обосновать формат строки токенов;

- разработать алгоритм и реализовать подпрограммы лексического анализа;

- разработать алгоритм и реализовать подпрограммы анализа конструкций языка;

- разработать тесты для тестирования программы;

- тестировать и отладить программу;

- оставить отчет по домашнему заданию;

- продемонстрировать работу программы преподавателю;

- защитить домашнее задание преподавателю.

Разработать грамматику и распознаватель выражений языка программирования Pascal, оперирующих вещественными числами в формате с фиксированной точкой. Например: -34.3456 + 0.56* 0.7989.

Выполнение

Грамматика в форме Бекуса-Наура представлена в листинге 1.

Листинг 1 – Грамматика в форме Бекуса-Наура

```
<знак>     ::= -|+
<цифра>    ::= 0|1|2|3|4|5|6|7|8|9
<оп>       ::= *|/|<знак>

<число>    ::= <чбз>|<чсз>
<чсз>      ::= <знак><чбз>
<чбз>      ::= <рц>.<рц>
<рц>       ::= <цифра>|<цифра><рц>

<выр>      ::= <выр_tail>|<число><оп><выр_tail>|<число>
<выр_tail>::= (<выр>)<оп><выр_tail>|(<выр>)|<чбз><оп><выр_tail>|<чбз>
```

Используемые сокращения:

- оп – операция;

- чсз – число со знаком;

- чбз – число без знака;

- рц – ряд цифр;

- выр – выражение;

- выр_tail – хвостовое выражение.

Построенные синтаксические диаграммы представлена на рисунках 1 и 2.

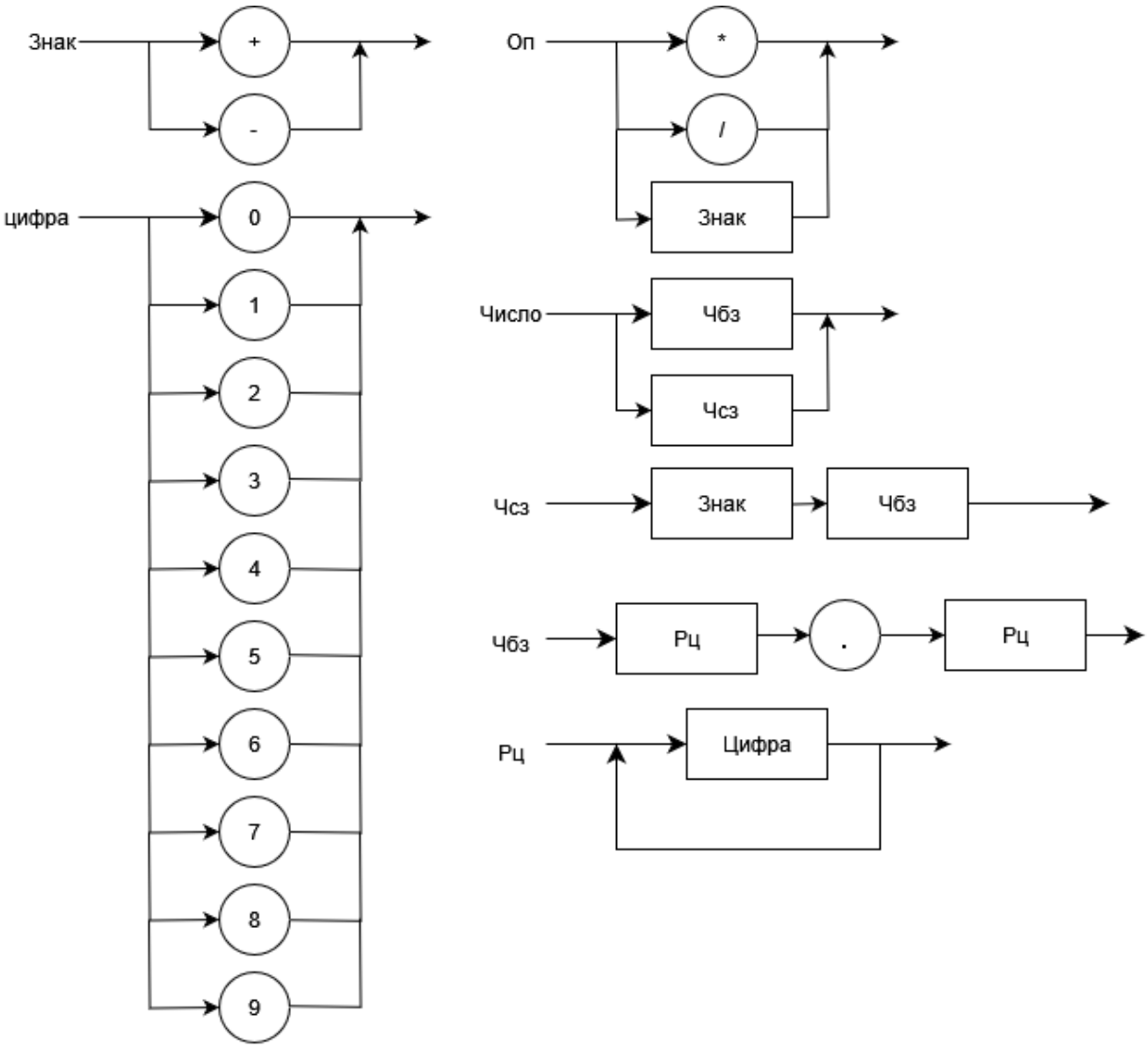

Рисунок 1 – Синтаксические диаграммы

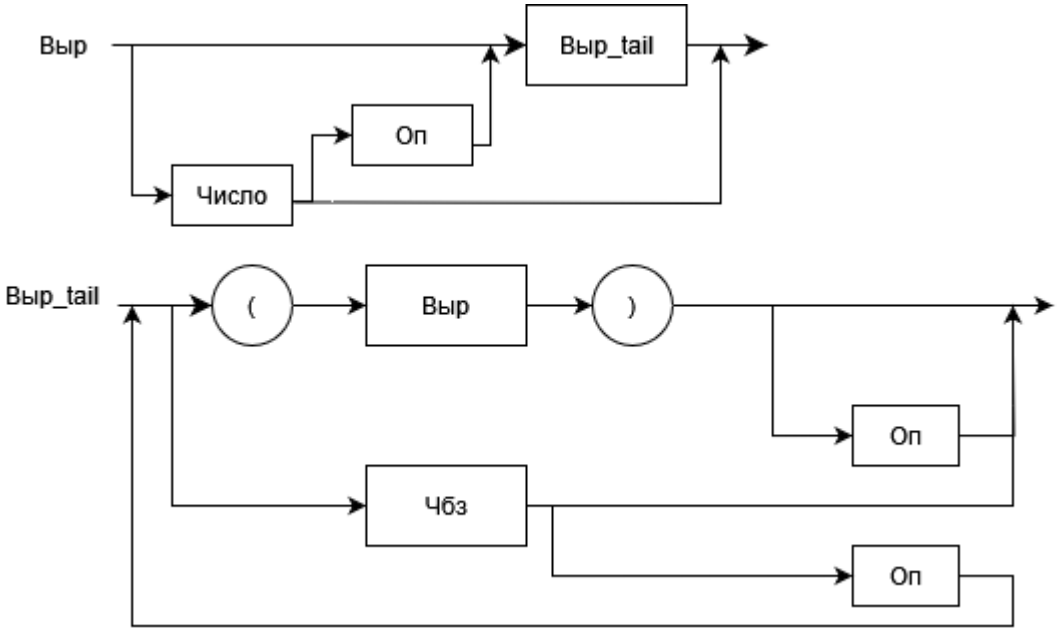

Рисунок 2 – Синтаксические диаграммы

Данная грамматика относится к контекстно-свободным грамматикам, то есть принадлежит к 2 типу, так как имеет вложенную рекурсию. Для реализации алгоритма для данной грамматики целесообразно использовать метод рекурсивного спуска.

Код программы представлен в листинге 2. Программа написана на языке Си.

Листинг 2 – Код распознавателя

```
#include <stdio.h>
#include <string.h>

// Разработать грамматику и распознаватель выражений языка
программирования Pascal, оперирующих вещественными числами в
формате с фиксированной точкой. Например:
// -34.3456 + 0.56* 0.7989

#define ASSERT_NOT_NULL(x) if (x == NULL) { return NULL; }
#define TRUE 1
#define FALSE 0

// <цифра> ::= 0|1|2|3|4|5|6|7|8|9
char * digit(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    if (inp[0] >= '0' && inp[0] <= '9') {
        return ++inp;
    } else {
        return NULL;
    }
}

// <знак>  ::= -|+
char * sign(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    if (inp[0] == '-' || inp[0] == '+') {
        if (print) {
            printf("Распознан знак: %.*s\n", 1, inp);
        }
        return ++inp;
    } else {
        if (print) {
            printf("Ошибка знака: %.*s\n", 1, inp);
        }
        return NULL;
    }
}

// <оп>   ::= *|/|<знак>
char * op(char * inp, int print) {
```

```
    ASSERT_NOT_NULL(inp);
    char * next;
    if (
        (next = inp, ++next, next[-1] == '*') ||
        (next = inp, ++next, next[-1] == '/') ||
        (next = sign(inp, FALSE))
    ) {
        if (print) {
            printf("Распознана операция: %.*s\n", 1, inp);
        }
        return next;
    } else {
        if (print) {
            printf("Ошибка операции: %.*s\n", 1, inp);
        }
        return NULL;
    }
}

// <рц>     ::= <цифра>|<цифра><рц>
char * digit_seq(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next;
    if (
        (next = digit_seq(digit(inp, FALSE), FALSE)) ||
        (next = digit(inp, FALSE))
    ) {
        return next;
    } else {
        return NULL;
    }
}

// <чбз>    ::= <рц>.<рц>
char * unumber(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next = inp;
    if (
        (next = digit_seq(inp, FALSE)) &&
        (++next, next[-1] == '.') &&
        (next = digit_seq(next, FALSE))
    ) {
        if (print) {
            printf("Распознано число без знака: %.*s\n",
(int)(strlen(inp)-strlen(next)), inp);
        }
        return next;
    } else {
        if (print) {
            printf("Ошибка числа без: %.*s\n", strlen(inp),
inp);
        }
        return NULL;
```

```
    }
}

// <чсз>    ::= <знак><чбз>
char * inumber(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next;
    if (next = unumber(sign(inp, print), print)) {
        return next;
    } else {
        return NULL;
    }
}

// <число>  ::= <чбз>|<чсз>
char * number(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next;
    if (
        (next = inumber(inp, print)) ||
        (next = unumber(inp, print))
    ) {
        return next;
    } else {
        return NULL;
    }
}

char * expr_tail(char * inp, int print);

// <выр>     ::= <выр_tail>|<число><оп><выр_tail>|<число>
char * expr(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next;
    if ((
        (next = expr_tail(inp, print))
    ) || (
        (next = number(inp, print)) &&
        (next = op(next, print)) &&
        (next = expr_tail(next, print))
    ) || (
        (next = number(inp, print))
    )) {
        return next;
    } else {
        return NULL;
    }
}

// <выр_tail>::=
(<выр>)<оп><выр_tail>|(<выр>)|<чбз><оп><выр_tail>|<чбз>
char * expr_tail(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
```

```
    char * next;
    if ((
        (next = inp, ++next, next[-1] == '(') &&
        (next = expr(next, print)) &&
        (++next, next[-1] == ')') &&
        (next = op(next, print)) &&
        (next = expr_tail(next, print))
    ) || (
        (next = inp, ++next, next[-1] == '(') &&
        (next = expr(next, FALSE)) &&
        (++next, next[-1] == ')')
    ) || (
        (next = unumber(inp, print)) &&
        (next = op(next, print)) &&
        (next = expr_tail(next, print))
    ) || (
        (next = unumber(inp, FALSE))
    )) {
        // printf("recognised: %.*s\n", (int)(strlen(inp)-
strlen(next)), inp);
        return next;
    } else {
        return NULL;
    }
}

char * parse(char * inp, int print) {
    ASSERT_NOT_NULL(inp);
    char * next = expr(inp, print);
    if (next != NULL && strlen(next) == 0) {
        printf("Распознано: %.*s\n", (int)(strlen(inp)-
strlen(next)), inp);
        printf("Конструкция распознана.\n");
    } else {
        printf("Обнаружена ошибка.\n");
    }
    return next;
}

void rem(char* s, char c) {
    for (int i = 0; i < strlen(s); i++)
        if (s[i] == '\n') {
            s[i] = '\0';
            break;
        }
    int j, n = strlen(s);
    for (int i = j = 0; i < n; i++)
        if (s[i] != c)
            s[j++] = s[i];

    s[j] = '\0';
}
```

```
int main() {
    char inp[100];
    while(1) {
        printf("Введите строку (end для выхода):\n");
        gets(inp);
        if (inp[0] == 'e' && inp[1] == 'n' && inp[2] == 'd') {
            return 0;
        }
        rem(inp, ' ');
        parse(inp, TRUE);
    }
    // parse("a\0", TRUE);
    return 0;
}
```

Тестирование

Результаты тестирования грамматики представлены в таблице 1.

Таблица 1 – Тестирование

№	Поток ввода	Ожидаемый результат	Фактический результат	Вердикт
1	-34.3456+0.56*0.7989	Распознано	Ошибка числа без знака: -34.3456+0.56*0.7989 Распознан знак: - Распознано число без знака: 34.3456 Распознана операция: + Распознано число без знака: 0.56 Распознана операция: * Распознано число без знака: 0.7989 Ошибка операции: Распознано: -34.3456+0.56*0.7989 Конструкция распознана.	Верно
2	-32.0*(-1.3*1.4)/(1.3)	Распознано	Ошибка числа без: -32.0*(-1.3*1.4)/(1.3) Распознан знак: -	Верно

			Распознано число без знака: 32.0 Распознана операция: * Ошибка числа без: - 1.3*1.4)/(1.3) Распознан знак: - Распознано число без знака: 1.3 Распознана операция: * Распознано число без знака: 1.4 Ошибка операции:) Распознана операция: / Распознано число без знака: 1.3 Ошибка операции:) Ошибка операции: Распознано: -32.0*(- 1.3*1.4)/(1.3) Конструкция распознана.	
3	a+b	Ошибка распознавания	Ошибка числа без знака: a+b Ошибка знака: a Ошибка числа без знака: a+b Ошибка знака: a Ошибка числа без знака: a+b Обнаружена ошибка.	Верно
4	1.3a4.1	Ошибка распознавания	Распознано число без знака: 1.3 Ошибка операции: a Обнаружена ошибка.	Верно

5	-0.1+0..1	Ошибка распознавания	Ошибка числа без знака: -0.1+0..1 Распознан знак: - Распознано число без знака: 0.1 Распознана операция: + Ошибка числа без знака: 0..1 Распознан знак: - Распознано число без знака: 0.1 Обнаружена ошибка.	Верно

Вывод

В результате выполнения домашнего задания были закреплены знания теоретических основ и основных методов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков. Также по заданию составлена грамматика в форме Бекуса-Наура для распознавателя выражений с вещественными числами, сделаны синтаксические диаграммы, определен тип грамматики и написана программа на языке Си по методу рекурсивного спуска.

Контрольные вопросы

1) Дайте определение формального языка и формальной грамматики.

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка: G = (VT, VN, P, S), где VT – алфавит языка или множество терминальных (незаменяемых) символов; VN – множество нетерминальных (заменяемых) символов – вспомогательный алфавит, символы которого обозначают допустимые понятия языка, $VT \cap VN = \emptyset$; $V = VT \cup VN$ – словарь грамматики; P – множество порождающих правил – каждое правило состоит из пары строк (α, β), где $\alpha \in V+$ – левая часть правила, β V* –

правая часть правила: α ∈ β, где строка α должна содержать хотя бы один нетерминал; S ∈ VN.

– начальный символ – аксиома грамматики.

Формальная грамматика, определяет правила допустимых конструкций языка.

2) Как определяется тип грамматики по Хомскому?

Тип 0 – грамматики фразовой структуры или грамматики «без ограничений»: α -> β, где α ∈ V+, β ∈ V*– в таких грамматиках допустимо наличие любых правил вывода, что свойственно грамматикам естественных языков.

Тип 1 – контекстно-зависимые (неукорачивающие) грамматики: α -> β, где α ∈ V+, β ∈ V*, | α | <= | β | – в этих грамматиках для правил вида α X β -> α x β возможность подстановки строки x вместо символа X определяется присутствием подстрок α и β, т. е. контекста, что также свойственно грамматикам естественных языков.

Тип 2 – контекстно-свободные грамматики: A -> β, где A ∈ VN, β ∈ V* – поскольку в левой части правила стоит нетерминал, подстановки не зависят от контекста.

Тип 3 – регулярные грамматики: A -> α, A -> αB, где A, B ∈ VN, α ∈ VT.

3) Поясните физический смысл и обозначения формы Бэкуса–Наура.

Форма Бэкуса-Наура (БНФ) связывает терминальные и нетерминальные символы, используя две операции: «::=» – «можно заменить на»; «|» – «или». Главный плюс этого - группировка правил, определяющих каждый нетерминал.

4) Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете? Лексический анализ – это первый этап процесса компиляции текста, написанного на формальном языке программирования. Сканированием преобразуется строка в набор символов. Каждый из которых представляет из себя токен.

5) Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?

Синтаксический анализ – процесс распознавания конструкций языка в строке токенов. Метод рекурсивного спуска для грамматик LL(k). Стековый метод для грамматик LR(k).

6) Что является результатом лексического анализа? Результатом лексического анализа является строка токенов, чаще всего записанных в виде таблицы.

7) Что является результатом синтаксического анализа?

Результатом, помимо распознавания заданной конструкции, является информация об ошибках в выражениях, операторах и описаниях программы.

8) В чем заключается метод рекурсивного спуска?

Метод рекурсивного спуска заключается в построении синтаксических диаграмм всех разбираемых конструкций, потом по этим диаграммам разработать функции проверки конструкций, а затем составить основную программу, начинающую вызов функций с функции, реализующей аксиому языка.

9) Что такое таблица предшествования и для чего она строится?

Она строится во время использования грамматики с предшествованием LR(k). Когда строка разбивается на набор символов, являющиеся токенами, строится таблицы предшествования. Она нужна для того, чтобы было видно какой результат следует ожидать от той, или иной операции (когда начинается и заканчивается “основа”).

10) Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Таблица использует специальные обозначении:

– ? – ошибка;

– < – начало основы;

– > – конец основы;

– () – скобки – принадлежат одной основе;

– ► – начало выражения;

– ◄ – конец выражения.

И когда идет проход по строке токенов, происходит сопоставление токенов со значениями в таблице, из чего определяется начало или конец основы или сигнал об ошибке.