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

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ_____
КАФЕДРА _____КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ_____

О Т Ч Е Т

по домашней работе № ___2__

Дисциплина: Машинно-зависимые языки и основы компиляции

Название домашней работы: Лексические и синтаксические анализаторы

Студент гр. **ИУ6-41Б** _____ _____
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2025

СОДЕРЖАНИЕ

Цель работы

Закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.

Задание

Разработать грамматику и распознаватель описания языка программирования Pascal, включающего оператор цикла-пока и оператор присваивания. Считать, что условие – значение переменной логического типа, тело цикла содержит не более одного оператора, а оператор присваивания в правой части содержит только идентификаторы или целые константы. Например:

while d do while ii do gyu:=5;

Решение

1) Составим грамматику конструкций в расширенной форме Бэкуса-Наура, результат представлен на листинге 1:

Листинг 1 — грамматика в расширенной форме Бэкуса-Наура

```
<программа> ::= <цикл>;

<цикл> ::= while <идентификатор> do <тело>

<тело> ::= <цикл> | <оператор>;

<оператор> ::= <идентификатор> := <целое число>;

<идентификатор> ::= <буква> | <идентификатор><буква> |
<идентификатор><цифра>

<буква> ::= a | b | ... | z | A | B | ... | Z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<целое число> ::= <цифра> | <цифра><целое число>
```

2) Составим синтаксическую диаграмму заданных грамматик, она изображена на рисунке 1.

Рисунок 1 — синтаксическая диаграмма

Поскольку слева стоит единственный нетерминал, грамматика не относится к грамматикам первого типа. Однако в ней присутствует правило только левосторонней рекурсии и нет вложенной. Следовательно, грамматика принадлежит классу грамматик третьего типа. Поэтому воспользуемся конечным автоматом. Таблица автомата представлена на таблице 1, где w — токен while, i — токен идентификатора, d — токен do, n – токен целого.

Таблица 1 – Таблица автомата

q\токен	w	i	d	n	:=	;	K
1	2						
2		3					
3			4				
4	1	5					
5					6		
6		7		7			
7							K

3) Разработаем программу в соответствии с вариантом. Ниже представлен код программы листингом 2.

Листинг 2 — код программы

```
#include <iostream>
#include <string>
#include <vector>
#include <regex>

enum TokenType {
    WHILE, IDENT, DO, ASSIGN, NUM, SEMICOLON, END, INVALID
};

struct Token {
    TokenType type;
    std::string value;
};

// Лексический анализатор
std::vector<Token> lexer(const std::string& input) {
    std::vector<Token> tokens;
    std::regex re(R"((while)|(do)|([a-zA-Z_]\w*)|(:=)|(-
?\d+)|(;))");
    auto words_begin = std::sregex_iterator(input.begin(),
input.end(), re);
    auto words_end = std::sregex_iterator();

    for (auto it = words_begin; it != words_end; ++it) {
        std::smatch match = *it;
        if (match[1].matched)
            tokens.push_back({ WHILE, "while" });
        else if (match[2].matched)
            tokens.push_back({ DO, "do" });
        else if (match[3].matched)
            tokens.push_back({ IDENT, match[3] });
        else if (match[4].matched)
            tokens.push_back({ ASSIGN, ":=" });
        else if (match[5].matched)
            tokens.push_back({ NUM, match[5] });
        else if (match[6].matched)
            tokens.push_back({ SEMICOLON, ";" });
        else
            tokens.push_back({ INVALID, match.str() });
    }

    tokens.push_back({ END, "" });
    return tokens;
```

Продолжение листинга 2

```
}

// Синтаксический анализатор
bool parse(const std::vector<Token>& tokens) {
    int state = 1;
    for (const auto& token : tokens) {
        switch (state) {
        case 1:
            if (token.type == WHILE) state = 2;
            else return false;
            break;
        case 2:
            if (token.type == IDENT) state = 3;
            else return false;
            break;
        case 3:
            if (token.type == DO) state = 4;
            else return false;
            break;
        case 4:
            if (token.type == WHILE) state = 2;
            else if (token.type == IDENT) state = 5;
            else return false;
            break;
        case 5:
            if (token.type == ASSIGN) state = 6;
            else return false;
            break;
        case 6:
            if (token.type == IDENT || token.type == NUM) state
= 7;
            else return false;
            break;
        case 7:
            if (token.type == SEMICOLON) state = 8;
            else return false;
            break;
        case 8:
            return true;
        }
    }
    return state == 8; // Проверяем, что закончили в состоянии
8 (корректное завершение)
}

Using namespace std;
int main() {
    setlocale(LC_ALL, "Russian");
```

Продолжение листинга 2

```
        string input;
        cout << "Введите строку для анализа: ";
        getline(cin, input);
        auto tokens = lexer(input);
        if (parse(tokens)) {
            cout << "Строка корректна по синтаксису." << endl;
        }
        else {
            cout << "Синтаксическая ошибка." << endl;
        }
        return 0;
}
```

Протестирую программу на четырех различных вариантах вводимой строки. Результат данного тестирования в таблице 2:

Таблица 2 — результаты тестирования

Исходные данные	Ожидаемый результат	Полученный результат
while d do while ii do gyu:=5;	Строка распознана	Строка распознана
while k dowhile d do while ii do gyu:=5;	Строка распознана	Строка распознана
while d do while ii do gyu:=5	Ошибка	Ошибка
while d do while ii do gyu=5;	Ошибка	Ошибка

Вывод

Выполняя данную лабораторную работу я закрепил знания теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.

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, $\beta \in V^*$ – правая часть правила: α ∈ β, где строка α должна содержать хотя бы один нетерминал; $S \in VN$ – 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 $\alpha \in V+$, $\beta \in V^*$– в таких грамматиках допустимо наличие любых правил вывода, что свойственно грамматикам естественных языков.

Тип 1 – контекстно-зависимые (неукорачивающие) грамматики: α -> β, где $\alpha \in V+$, $\beta \in V^*$, $|\alpha| <= |\beta|$ – 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 $A \in V_N$, $\beta \in V^*$ – поскольку в левой части правила стоит нетерминал, подстановки не зависят от контекста.

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

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 LR(k). Стековый метод для грамматик LL(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 LL(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.