Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ
к лабораторной работе №2
на тему

# ЛЕКСИЧЕСКИЙ АНАЛИЗ

Выполнил: студент гр.253502 Дриневский К.В.

Проверил: ассистент кафедры информатики
Гриценко Н.Ю.

Минск 2025

**СОДЕРЖАНИЕ**

# 1 ЦЕЛЬ РАБОТЫ

Цель данной лабораторной работы заключается в разработке и реализации лексического анализатора, который способен обрабатывать исходный код на языке Java. В рамках работы поставлены следующие задачи:

1 Изучение основ лексического анализа. Понять принципы работы лексического анализатора, его роль в компиляции и интерпретации языков программирования[1].

2 Разработка токенизатора. Реализовать программу, которая считывает исходный код и разбивает его на токены, распознавая различные типы, такие как ключевые слова, идентификаторы, числа, строковые и символьные литералы, операторы и ошибки[2].

3 Создание структуры токена. Определить структуру токена, содержащую информацию о типе токена, его значении, номере строки и уникальном идентификаторе.

4 Обработка входных данных. Реализовать функциональность для чтения исходного кода из файла и вывода результатов токенизации в другой файл, что позволяет удобно анализировать полученные токены.

5 Тестирование и отладка. Проверить работу лексического анализатора на различных примерах исходного кода, включая корректные и некорректные конструкции, для оценки его устойчивости и точности.

## 2 ХОД РАБОТЫ

В ходе выполнения лабораторной работы реализован синтаксический анализатор *Java* на *C++*. Листинг программного и тестового кодов приведен в приложении А. Он распознает такие типы токенов, как ключевые слова, идентификаторы, целые числа, числа с плавающей точкой, строки, символы, операторы.

На рисунке 2.1 представлено содержание файла с выходными данными.

```
 4   Token: OPERATOR Lexem: "{" Line: 1 Id: 4
 5   Token: KEYWORD Lexem: "public" Line: 2 Id: 1
 6   Token: KEYWORD Lexem: "static" Line: 2 Id: 5
 7   Token: KEYWORD Lexem: "void" Line: 2 Id: 6
 8   Token: IDENTIFIER Lexem: "main" Line: 2 Id: 7
 9   Token: OPERATOR Lexem: "(" Line: 2 Id: 8
10   Token: KEYWORD Lexem: "String" Line: 2 Id: 9
11   Token: OPERATOR Lexem: "[" Line: 2 Id: 10
12   Token: OPERATOR Lexem: "]" Line: 2 Id: 11
13   Token: IDENTIFIER Lexem: "args" Line: 2 Id: 12
14   Token: OPERATOR Lexem: ")" Line: 2 Id: 13
15   Token: OPERATOR Lexem: "{" Line: 2 Id: 4
```

Рисунок 2.1 – Результат выполнения

В тестовом коде Java было сделано 4 лексические ошибки. Введен символ “ф”. Анализатор обрабатывает этот случай корректно(рисунок 2.2).

```
Token: NUMBER Lexem: "123212" Line: 8 Id: 32
Token: ERROR Lexem: "" Line: 8 Id: 33
Token: ERROR Lexem: "" Line: 8 Id: 34
Token: NUMBER Lexem: "1252" Line: 8 Id: 35
```

Рисунок 2.2 – Символ “ф” в тестовом коде

Введен литерал “-3.422.5d”. Анализатор обрабатывает этот случай корректно(рисунок 2.3).

```
Token: ERROR Lexem: "-3.422" Line: 15 Id: 53
Token: FLOAT_NUMBER Lexem: ".5d" Line: 15 Id: 54
```

Рисунок 2.3 – Литерал “-3.422.5d” в тестовом коде

Введен литерал “4e”. Анализатор обрабатывает этот случай корректно(рисунок 2.4).


Рисунок 2.4 – Литерал “4e” в тестовом коде

Введен недопустимый символ “4e”. Анализатор обрабатывает этот случай корректно(рисунок 2.5).

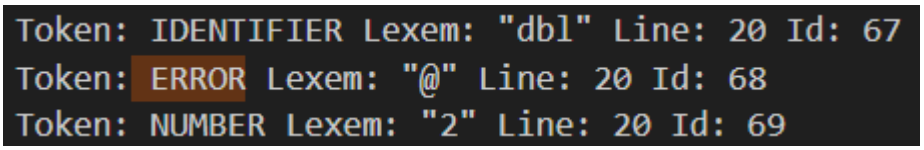


Рисунок 2.5 – Символ “@” в тестовом коде

Для каждого токена отображается его тип, соответствующая лексема, строка, в которой он находится,  и идентификатор. Если токен встречается повторно, то ему присваивается идентификатор его первого нахождения.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы был успешно реализован лексический анализатор подмножества языка Java, определенного в лабораторной работе №1, с использованием языка программирования C++. Разработка лексера позволила глубже понять принципы лексического анализа и его важность в процессе компиляции программного кода.

Лексический анализатор демонстрирует корректную работу, успешно обрабатывая различные типы токенов, такие как ключевые слова, идентификаторы, числовые и строковые литералы, а также операторы. Реализована функциональность для обработки некорректных лексем, что позволяет анализатору выявлять и сообщать об ошибках. Это важное дополнение, так как оно помогает быстро находить и исправлять ошибки в коде.

Результаты работы лексического анализатора были документированы, что позволяет легко отслеживать выдаваемые токены и их характеристики. Выводы и примеры, представленные в отчете, служат наглядным подтверждением успешности выполнения поставленных задач.

Таким образом, реализация данного лексического анализатора не только углубила теоретические знания, но и предоставила практический опыт разработки программного обеспечения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] GeeksForGeeks [Электронный ресурс]. – Режим доступа: https://translated.turbopages.org/proxy_u/en-ru.ru.9258fdd3-67a9de31-daa5d889-74722d776562/https/www.geeksforgeeks.org/lexical-analyzer-in-cpp/. – Дата доступа: 10.02.2025.

[2] Habr [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/713434/. – Дата доступа: 10.02.2025.

# ПРИЛОЖЕНИЕ А
## (обязательное)
## Листинг программного кода

Листинг 1 – main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <cctype>
#include <sstream>
enum TokenType {
    KEYWORD,
    IDENTIFIER,
    NUMBER,
    FLOAT_NUMBER,
    STRING_LITERAL,
    CHAR_LITERAL,
    OPERATOR,
    UNKNOWN,
    ERROR
};
struct Token {
    TokenType type;
    std::string value;
    int line;
    int id;

    std::string typeToString() const {
        switch (type) {
            case KEYWORD: return "KEYWORD";
            case IDENTIFIER: return "IDENTIFIER";
            case NUMBER: return "NUMBER";
            case FLOAT_NUMBER: return "FLOAT_NUMBER";
            case STRING_LITERAL: return "STRING_LITERAL";
            case CHAR_LITERAL: return "CHAR_LITERAL";
            case OPERATOR: return "OPERATOR";
            case ERROR: return "ERROR";
            default: return "UNKNOWN";
        }
    }
};
const std::unordered_set<std::string> javaKeywords = {
     "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char",
"class",  "const",  "continue",  "default",  "do",  "double",  "else",  "enum",
"extends",  "final",  "finally",  "float",  "for",  "goto",  "if",  "implements",
"import",  "instanceof",  "int",  "interface",  "long",  "native",  "new",  "null",
"package",  "private",  "protected",  "public",  "return",  "short",  "static",
"strictfp",  "super",  "switch",  "synchronized",  "this",  "throw",  "throws",
"transient",  "try",  "void",  "volatile",  "while",  "true",  "false",  "String",
"ArrayList", "HashMap", "HashSet"};
const std::unordered_set<std::string> javaOperators = {
      "+",  "-",  "*",  "/",  "%",  "++",  "--",  "==",  "!=",  ">",  "<",  ">=",  "<=",
"&&",  "||",  "!",  "=",  "+=",  "-=",  "*=",  "/=",  "%=",  "&",  "|",  "^",  "~",  "<<",
">>",  ">>>",  "?",  ":",  "::",  ".",  ",",  ";",  "(",  ")",  "{",  "}",  "[",  "]"};
class Lexer {
public:
    explicit Lexer(const std::string& sourceCode)
        : source(sourceCode), pos(0), line(1), nextTokenId(1) {}
    std::vector<Token> tokenize() {
        tokens.clear();
```

```
        while (pos < source.length()) {
            char currentChar = source[pos];
            if (std::isspace(currentChar)) {
                if (currentChar == '\n') {
                    line++; // Увеличиваем номер строки
                }
                pos++; // Пропускаем пробелы
            } else if (std::isalpha(currentChar) || currentChar == '_') {
                tokens.push_back(consumeIdentifierOrKeyword());
            } else if (std::isdigit(currentChar) ||
                       (currentChar == '.' && pos + 1 < source.length() &&
std::isdigit(source[pos + 1]))) {
                tokens.push_back(consumeNumber());
            } else if (currentChar == '"') {
                tokens.push_back(consumeStringLiteral());
            } else if (currentChar == '\'') {
                tokens.push_back(consumeCharLiteral());
              } else if (currentChar == '/' && (pos + 1 < source.length() &&
(source[pos + 1] == '/' || source[pos + 1] == '*'))) {
                consumeComment(); // Игнорируем комментарии
            } else if (isOperator(std::string(1, currentChar))) {
                tokens.push_back(consumeOperator());
            } else {
          tokens.push_back(createToken(ERROR, std::string(1, currentChar)));
                pos++;
            }
        }
        return tokens;
    }

private:
    std::string source;
    size_t pos;
    int line;
    int nextTokenId;
    std::vector<Token> tokens;
    std::unordered_map<std::string, int> tokenIds; // Хранение ID токенов
    bool isOperator(const std::string& op) {
        return javaOperators.find(op) != javaOperators.end();
    }
    Token createToken(TokenType type, const std::string& value) {
        if (tokenIds.find(value) == tokenIds.end()) {
            tokenIds[value] = nextTokenId++;
        }
        return {type, value, line, tokenIds[value]};
    }
    Token consumeIdentifierOrKeyword() {
        size_t start = pos;
              while (pos < source.length() && (std::isalnum(source[pos]) ||
source[pos] == '_')) {
            pos++;
        }
        std::string word = source.substr(start, pos - start);
            return createToken(javaKeywords.find(word) != javaKeywords.end() ?
KEYWORD : IDENTIFIER, word);
    }
    Token consumeNumber() {
        size_t start = pos;
        bool isFloat = false;
        if (pos > 0 && (source[pos - 1] == '-' || source[pos - 1] == '+') &&
            (pos - 1 == 0 || std::isspace(source[pos - 2]) || source[pos - 2]
== '=')) {
            tokens.pop_back();
            start = pos - 1;
        }
```

```
            while (pos < source.length() && (std::isdigit(source[pos]) ||
source[pos] == '.')) {
            if (source[pos] == '.') {
                if (isFloat) {
                return createToken(ERROR, source.substr(start, pos - start));
                }
                isFloat = true;
            }
            pos++;
        }
           if (pos < source.length() && (source[pos] == 'e' || source[pos] ==
'E')) {
            isFloat = true;
            pos++;
             if (pos < source.length() && (source[pos] == '+' || source[pos] ==
'-')) {
                pos++;
            }
            if (pos >= source.length() || !std::isdigit(source[pos])) {
                return createToken(ERROR, source.substr(start, pos - start));
            }
            while (pos < source.length() && std::isdigit(source[pos])) {
                pos++;
            }
        }
         if (pos < source.length() && (source[pos] == 'f' || source[pos] == 'F'
|| source[pos] == 'd' || source[pos] == 'D')) {
            isFloat = true;
            pos++;
        }
      return createToken(isFloat ? FLOAT_NUMBER : NUMBER, source.substr(start,
pos - start));
    }
    Token consumeStringLiteral() {
        size_t start = pos;
        pos++;
        while (pos < source.length() && source[pos] != '"') {
            if (source[pos] == '\\' && pos + 1 < source.length()) {
                pos += 2;
            } else {
                pos++;
            }
        }
        if (pos < source.length() && source[pos] == '"') {
            pos++;
                 return createToken(STRING_LITERAL, source.substr(start, pos -
start));
        }
        return createToken(ERROR, source.substr(start, pos - start));
    }
    Token consumeCharLiteral() {
        size_t start = pos;
        pos++;
        if (pos < source.length() && source[pos] == '\\') {
            pos += 2;
        } else {
            pos++;
        }
        if (pos < source.length() && source[pos] == '\'') {
            pos++;
                   return createToken(CHAR_LITERAL, source.substr(start, pos -
start));
        }
        return createToken(ERROR, source.substr(start, pos - start));
    }
    void consumeComment() {
```

```
        if (source[pos + 1] == '/') {
            pos += 2;
            while (pos < source.length() && source[pos] != '\n') {
                pos++;
            }
        } else if (source[pos + 1] == '*') {
            pos += 2;
                 while (pos + 1 < source.length() && !(source[pos] == '*' &&
source[pos + 1] == '/')) {
                pos++;
            }
            if (pos + 1 < source.length()) {
                pos += 2;
            }
        }
    }
    Token consumeOperator() {
        size_t start = pos;
          while (pos < source.length() && isOperator(source.substr(start, pos -
start + 1))) {
            pos++;
        }
        return createToken(OPERATOR, source.substr(start, pos - start));
    }
};
int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cerr << "Usage: " << argv[0] << " <source_file>\n";
        return 1;
    }
    std::ifstream inputFile(argv[1]);
    if (!inputFile) {
        std::cerr << "Error: Could not open file " << argv[1] << "\n";
        return 1;
    }
    std::string sourceCode((std::istreambuf_iterator<char>(inputFile)),
                           std::istreambuf_iterator<char>());
    Lexer lexer(sourceCode);
    std::vector<Token> tokens = lexer.tokenize();
                                    std::string        output_file_name        =
"D:\\Study\\6_semestr\\MTran\\Lab2\\output.txt";
    std::ofstream outputFile(output_file_name);
    if (!outputFile) {
        std::cerr << "Error: Could not open output file\n";
        return 1;
    }
    for (const Token& token : tokens) {
        outputFile << "Token: " << token.typeToString()
                   << " Lexem: \"" << token.value << "\""
                   << " Line: " << token.line
                   << " Id: " << token.id << "\n";
    }
    std::cout << "Tokens written to output.txt\n";
    return 0;
}
```

Листинг 2 –TestLexer.java

```
public class TestLexer {
    public static void main(String[] args) {
        // Пример переменных
        byte byte1 = 109;
        short short1 = -129&00;
        int number1 = 42;
        int number2 = -12;
        long long1 = 123212ф1252;
```

```
        String text = "Hello, World!";
        char letter = 'A';
        boolean isValid = true;
        float fl1 = 3.4f;
        float fl2 = -3.41F;
        double dbl7 = -3.422d;
        double dbl3 = -3.422.5d;
        double dbl6 = -3.4225D;
        double dbl4 = 3o9.45ee;
        double dbl1 = 4e-2;
        double dbl5 = 4e;
        double dbl@2 = 4E+17;

        // Условие
        if (isValid) {
            System.out.println(text);
        } else {
            System.out.println("Invalid");
        }

        // Цикл
        for (int i = 0; i < number1; i++) {
            System.out.println(i);
        }
        number1++;
        ++number2;
        ++4;
        --number1;
        number1--;
        // Массив
        int[] array = {1, 2, 3, 4, 5};
        for (int num : array) {
            System.out.println(num);
        }

        // Класс
        class InnerClass {
            void display() {
                System.out.println("Inside Inner Class");
            }
        }
        hg543
        // Создание объекта
        InnerClass inner = new InnerClass();
        inner.display();
    }
}
```