МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной техники

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №4

«Лексический анализатор»

по дисциплине
«Теория формальных языков»

Выполнил студент группы ИКБО-42-23 *Туляшева А.Т.*

Принял старший преподаватель *Боронников А.С.*

Практическая работа выполнена «__»_____2024 г.

«Зачтено» «__»_____2024 г.

Москва 2024

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Условие задачи: на выбранном языке программирования реализовать лексический анализатор, согласно варианту КР.

Согласно 9 варианту курсовой работы грамматика языка включает следующие синтаксические конструкции:

<операции_группы_отношения>::= <> | = | < | <= | > | >=

<операции_группы_сложения>::= + | - | or

<операции_группы_умножения>::= * | / | and

<унарная_операция>::= not

<программа> = {/ (<описание> | <оператор>) (: | переход строки) /} end

<описание>::= {<идентификатор> {, <идентификатор> } : <тип> ;}

<тип>::= % | ! | $

<оператор>::= <составной> | <присваивания> | <условный> | <фиксированного_цикла> | <условного_цикла> | <ввода> | <вывода>

<составной>::= «[» <оператор> { (: | перевод строки) <оператор> } «]»

<присваивания>::= <идентификатор> as <выражение>

<условный>::= if <выражение> then <оператор> [else <оператор>]

<фиксированного_цикла>::= for <присваивания> to <выражение> do <оператор>

<условного_цикла>::= while <выражение> do <оператор>

<ввода>::= read «(»<идентификатор> {, <идентификатор> } «)»

<вывода>::= write «(»<выражение> {, <выражение> } «)»

Многострочные комментарии в программе { … }

Выражения языка задаются правилами:

<выражение>::=<операнд>{<операции_группы_отношения><операнд>}

<операнд>::=<слагаемое>{<операции_группы_сложения><слагаемое>}

<слагаемое>::=<множитель>{<операции_группы_умножения><множитель>}

<множитель>::= <идентификатор> | <число> | <логическая_константа>

<число>::= <целое> | <действительное>

<логическая_константа>::= true | false

Правила, определяющие идентификатор, букву и цифру:

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

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

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

Правила, определяющие целые числа:

<целое>::= <двоичное> | <восьмеричное> | <десятичное> |

<шестнадцатеричное>

<двоичное>::= {/ 0 | 1 /} (B | b)

<восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (O | o)

<десятичное>::= {/ <цифра> /} [D | d]

<шестнадцатеричное>::= <цифра> {<цифра> | A | B | C | D | E | F | a | b | c |

Правила, описывающие действительные числа:

<действительное>::= <числовая_строка> <порядок> |

[<числовая_строка>] . <числовая_строка> [порядок]

<числовая_строка>::= {/n<цифра> /}

<порядок>::= (E | e)[+ | -] <числовая_строка>

Здесь для записи правил грамматики используется форма Бэкуса-Наура (БНФ). В записи БНФ левая и правая части порождения разделяются символом “::=”, нетерминалы заключены в угловые скобки, а терминалы – просто символы, используемые в языке.

РАЗРАБОТКА ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА

Лексический анализатор – подпрограмма, которая принимает на вход исходный текст программы и выдает последовательность *лексем* – минимальных элементов программы, несущих смысловую нагрузку.

В модельном языке программирования выделяют следующие типы лексем:

- ключевые слова;
- ограничители;
- числа;
- идентификаторы.

При разработке лексического анализатора, ключевые слова и ограничителя известны заранее, идентификаторы и числовые константы – вычисляются в момент разбора исходного текста.

Для каждого типа лексем предусмотрена отдельная таблица. Таким образом, внутреннее представление лексемы – пара чисел (n, k), где n – номер таблицы лексем, k – номер лексемы в таблице.

Кроме того, в исходном коде программы кроме ключевых слов, идентификаторов и числовых констант может находиться произвольное число пробельных символов («пробел», «табуляция», «перенос строки», «возврат каретки») и комментариев, заключенных в фигурные скобки.

Лексический анализ текста проводится по регулярной грамматике. Известно, что регулярная грамматика эквивалентна конченому автомату, следовательно, для написания лексического анализатора необходимо построить диаграмму состояний, соответствующего конечного автомата (рис. 1).

Код лексического анализатора приведен в листинге 1.

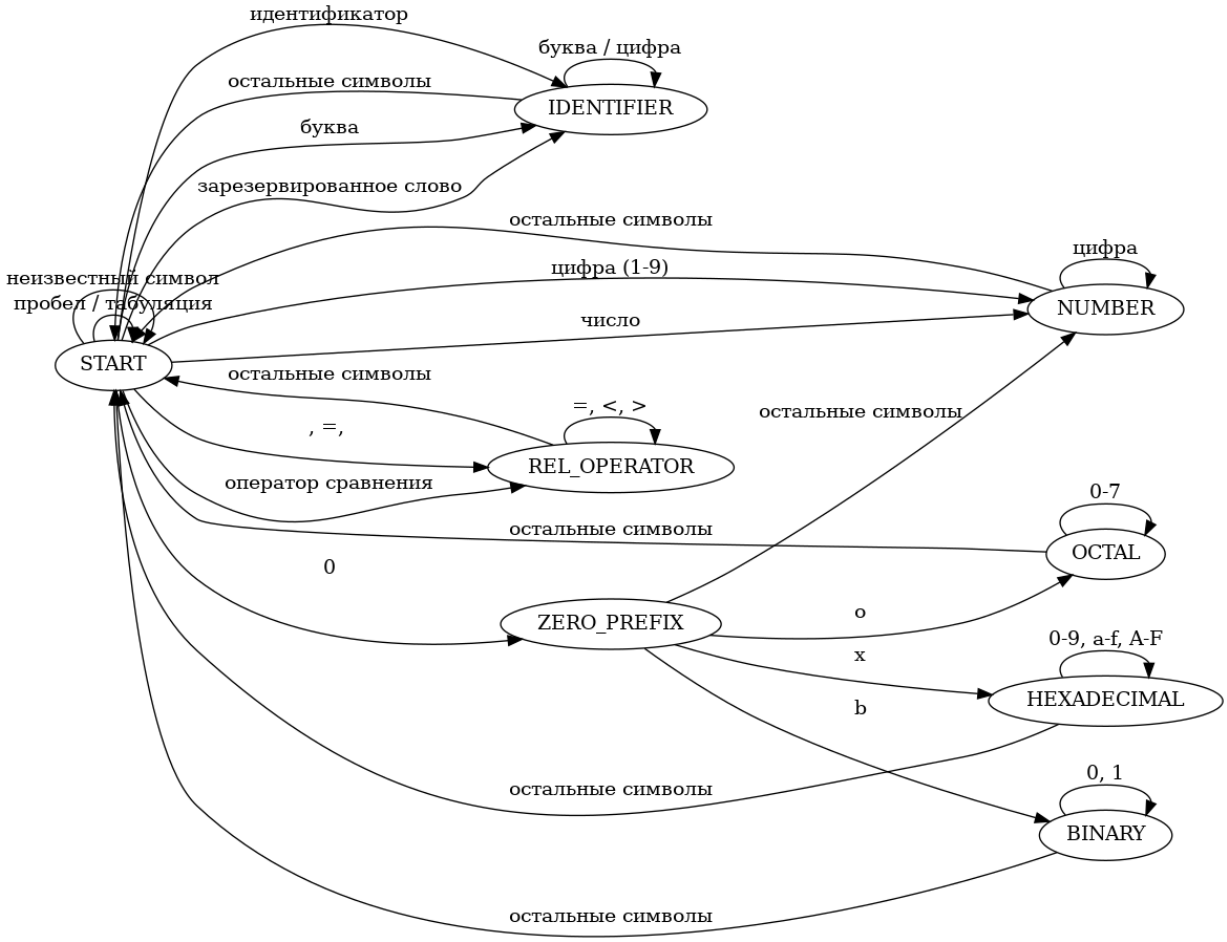

Рисунок 1 – Диаграмма состояний лексического анализатора

КОД ПРОГРАММЫ

Для реализации лексического анализатор был написал класс Lexer, куда входят списки ключевых слов, разделителей, типов данных и операторов.

Листинг 1 – лексический анализатор

```
class Lexer:
    RESERVED_WORDS = {  # ключевые слова
        "if": "IF",
        "then": "THEN",
        "else": "ELSE",
        "for": "FOR",
        "to": "TO",
        "do": "DO",
        "while": "WHILE",
        "read": "READ",
        "write": "WRITE",
        "as": "ASSIGN",
        "true": "TRUE",
        "false": "FALSE",
        "and": "AND",
        "or": "OR",
        "not": "NOT",
    }
```

Продолжение листинга 1

```
    SYMBOLS = {  # разделители
        "{": "LBRACE",
        "}": "RBRACE",
        "[": "LBRACKET",
        "]": "RBRACKET",
        ";": "SEMICOLON",
        ",": "COMMA",
        "(": "LPAREN",
        ")": "RPAREN",
        ":": "COLON",
    }

    DATA_TYPES = {  # типы данных
        "%": "INT",
        "!": "REAL",
        "$": "BOOL",
    }

    OPERATORS = {  # операторы
        "+": "ADD_OP",
        "-": "ADD_OP",
        "*": "MUL_OP",
        "/": "MUL_OP",
        "=": "REL_OP",
        "<>": "REL_OP",
        ">=": "REL_OP",
        "<=": "REL_OP",
        "<": "REL_OP",
        ">": "REL_OP",
    }

    def __init__(self, program_text):
        self.program_lines = program_text.splitlines() # текст разбивается на
строки
        self.lexemes = []  # список лексем
        self.column_number = 0  # текущая колонка
        self.line_number      =      0            #      текущая      строка
 def analyze_line(self, line):  # анализ одной строки
        accumulator = ""   # символы лексемы
        current_state = "START"  # начальное состояние
        for idx, char in enumerate(line, start=1):
            self.column_number = idx  # номер колонки

            if current_state == "START":
                if char.isspace():  # пропуск пробелов
                    continue
                elif char.isalpha():
                    accumulator += char
                    current_state = "IDENTIFIER"  # обработка идентификаторов
                elif char.isdigit() and char != '0':
                    accumulator += char
                    current_state = "NUMBER"  # обработка чисел
                elif char in self.SYMBOLS:
                    self.add_lexeme(self.SYMBOLS[char],  char)    #  добавление
символа
                elif char in "<>=":
                    accumulator += char
                    current_state = "REL_OPERATOR"  # операторы сравнения
                elif char in self.OPERATORS:  # добавление оператора
                    self.add_lexeme(self.OPERATORS[char], char)
                elif char in self.DATA_TYPES:  # добавление типа данных
```

Продолжение листинга 1

```
                    self.add_lexeme(self.DATA_TYPES[char], char)
                elif char == '0':
                    accumulator += char
                    current_state = "ZERO_PREFIX"  # числа с префиксами
                else:
                    print(f"Ошибка: Неизвестный символ '{char}' в строке
{self.line_number}, колонке {self.column_number}.")

            elif current_state == "IDENTIFIER":  # обработка идентификаторов
                if char.isalnum():
                    accumulator += char
                else:
                    if accumulator in self.RESERVED_WORDS:
                        self.add_lexeme(self.RESERVED_WORDS[accumulator],
accumulator)
                    else:
                        self.add_lexeme("IDENTIFIER", accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)   # повторный вызов для текущего
символа
            # аналогично для других состояний:
            elif current_state == "NUMBER":
                if char.isdigit():
                    accumulator += char
                else:
                    self.add_lexeme("NUMBER", accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)

            elif current_state == "REL_OPERATOR":
                if char in "=>":
                    accumulator += char
                    if accumulator in self.OPERATORS:
                        self.add_lexeme(self.OPERATORS[accumulator],
accumulator)
                        accumulator = ""
                        current_state = "START"
                else:
                    if accumulator in self.OPERATORS:
                        self.add_lexeme(self.OPERATORS[accumulator],
accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)

            elif current_state == "ZERO_PREFIX":
                if char.lower() in "box":
                    accumulator += char
                    if char.lower() == 'b':
                        current_state = "BINARY"
                    elif char.lower() == 'o':
                        current_state = "OCTAL"
                    elif char.lower() == 'x':
                        current_state = "HEXADECIMAL"
                else:
                    current_state = "NUMBER"
                    self.analyze_line(char)

            elif current_state == "BINARY":
```

Продолжение листинга 1

```
                if char in '01':
                    accumulator += char
                else:
                    self.add_lexeme("NUMBER", accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)

            elif current_state == "OCTAL":
                if char in '01234567':
                    accumulator += char
                else:
                    self.add_lexeme("NUMBER", accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)

            elif current_state == "HEXADECIMAL":
                if char in '0123456789abcdefABCDEF':
                    accumulator += char
                else:
                    self.add_lexeme("NUMBER", accumulator)
                    accumulator = ""
                    current_state = "START"
                    self.analyze_line(char)
        # завершаем обработку строки, если что-то осталось еще
        if current_state == "IDENTIFIER" and accumulator:
            if accumulator in self.RESERVED_WORDS:
                self.add_lexeme(self.RESERVED_WORDS[accumulator], accumulator)
            else:
                self.add_lexeme("IDENTIFIER", accumulator)

    def add_lexeme(self, lexeme_type, lexeme_value):   # сохраняет лексемы в
список
        self.lexemes.append((lexeme_type, lexeme_value))

    def display_lexemes(self):  # вывод лексем

       for lexeme in self.lexemes:
        self.optional_semicolon()   # Проверяем наличие необязательной точки с
запятой
            print(lexeme)

    def execute(self):  # запуск анализа строк
        for line_idx, line in enumerate(self.program_lines, start=1):
            self.line_number = line_idx  # обновление номера строки
            self.analyze_line(line)  # анализ строки

    def retrieve_lexemes(self):
        return self.lexemes  # список лексем
```

ТЕСТИРОВАНИЕ

В отдельном текстовом файле представлен код программы, соответствующий синтаксису языка. В результате лексического анализа выводятся токены.

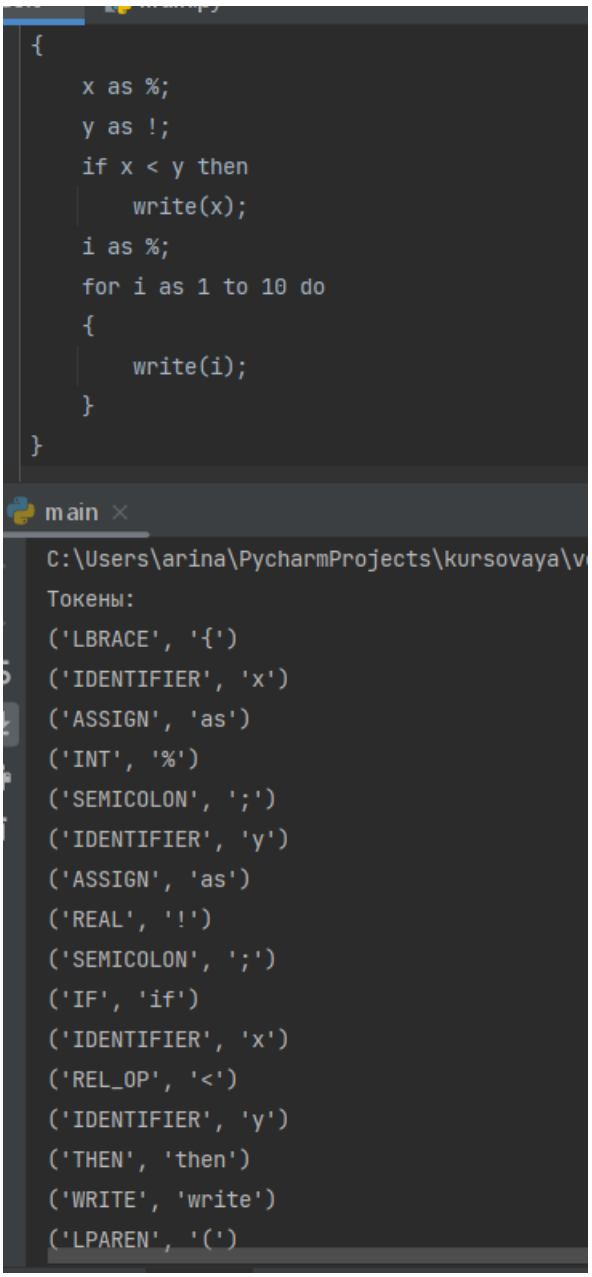

Рисунок 2 - Тестирование

Токены выводятся верно. Тестирование успешно.