



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт информационных технологий  
Кафедра вычислительной техники

## КУРСОВАЯ РАБОТА

по дисциплине

Теория формальных языков

(наименование дисциплины)

Тема курсовой работы

Разработка распознавателя модельного языка  
программирования (вариант №18)

(наименование темы)

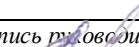
Студент группы ИКБО-41-23  
(учебная группа)

Киселев Д.А.  
(Фамилия И.О.)

  
(подпись студента)

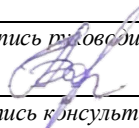
Руководитель  
курсовой работы доцент каф. ВТ, к.т.н.

Унгер А.Ю.

  
(подпись руководителя)

Консультант ст. преп. каф. ВТ

Боронников А.С.

  
(подпись консультанта)

Работа представлена к защите « » \_\_\_\_\_ 2024 г.

Допущен к защите « » \_\_\_\_\_ 2024 г.

Москва 2024



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

Институт информационных технологий  
Кафедра вычислительной техники

**Утверждаю**

Заведующий кафедрой \_\_\_\_\_

(подпись)

Платонова О.В.

«24» сентября 2024 г.

## **ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

« \_\_\_\_\_ Теория формальных языков \_\_\_\_\_ »

Студент Киселев Даниил Андреевич Группа ИКБО-41-23

Тема работы: Разработка распознавателя модельного языка программирования

Исходные данные: Грамматика модельного языка согласно варианту №18,  
язык программирования – Python

Перечень вопросов, подлежащих разработке, и обязательного графического материала: \_\_\_\_\_

- 1) Проектирование диаграммы состояний лексического анализатора;
- 2) Разработка лексического анализатора;
- 3) Разработка синтаксического анализатора;
- 4) Разработка семантического анализатора;
- 5) Описание спецификации основных процедур и функций;
- 6) Исходный код с комментариями;
- 7) Тестирование распознавателя модельного языка программирования.

Срок представления к защите курсовой работы: до « 23 » декабря 2024 г.

Задание на курсовую работу выдал \_\_\_\_\_ ( Боронников А.С. )  
Подпись ФИО консультанта

Задание на курсовую работу получил «19» сентября 2024 г.  
\_\_\_\_\_ ( Киселев Д.А. )  
Подпись ФИО обучающегося

Москва 2024

# СОДЕРЖАНИЕ

СОДЕРЖАНИЕ .....	3
ВВЕДЕНИЕ.....	4
1 ПОСТАНОВКА ЗАДАЧИ .....	5
2 ПОРЯДОК ВЫПОЛНЕНИЯ.....	7
3 ГРАММАТИКА МОДЕЛЬНОГО ЯЗЫКА .....	8
4 РАЗРАБОТКА ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА.....	11
5 ТЕСТИРОВАНИЕ ПРОГРАММЫ.....	13
ЗАКЛЮЧЕНИЕ .....	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	15
ПРИЛОЖЕНИЯ .....	17

# ВВЕДЕНИЕ

Несмотря на более чем полувековую историю вычислительной техники, рождение теории формальных языков ведет отсчет с 1957 года. В этот год американский ученый Джон Бэкус разработал первый компилятор языка Фортран. Он применил теорию формальных языков, во многом опирающуюся на работы известного ученого-лингвиста Н. Хомского – автора классификации формальных языков. Хомский в основном занимался изучением естественных языков, Бэкус применил его теорию для разработки языка программирования. Это дало толчок к разработке сотен языков программирования.

Несмотря на наличие большого количества алгоритмов, позволяющих автоматизировать процесс написания транслятора для формального языка, создание нового языка требует творческого подхода. В основном это относится к синтаксису языка, который, с одной стороны, должен быть удобен в прикладном программировании, а с другой, должен укладываться в область контекстно-свободных языков, для которых существуют развитые методы анализа.

Основы теории формальных языков и практические методы разработки распознавателей формальных языков составляют неотъемлемую часть образования современного инженера-программиста.

**Целью** данной курсовой работы является:

- освоение основных методов разработки распознавателей формальных языков на примере модельного языка программирования;

- приобретение практических навыков написания транслятора языка программирования;

- закрепление практических навыков самостоятельного решения инженерных задач, умения пользоваться справочной литературой и технической документацией.

# 1 ПОСТАНОВКА ЗАДАЧИ

**Разработать распознаватель модельного языка программирования согласно заданной формальной грамматике.**

Распознаватель – это специальный алгоритм, который позволяет определить принадлежность цепочки символов некоторому языку.

Распознаватель можно схематично представить в виде совокупности входной ленты, читающей головки, которая указывает на очередной символ на ленте, устройства управления (УУ) и дополнительной памяти (стек).

Конфигурацией распознавателя является:

- состояние УУ;
- содержимое входной ленты;
- положение читающей головки;
- содержимое дополнительной памяти (стека).

Трансляция исходного текста программы происходит в несколько этапов. Основными этапами являются следующие:

- лексический анализ;
- синтаксический анализ;
- семантический анализ;
- генерация целевого кода.

Лексический анализатор (ЛА) – это распознаватель, который группирует символы, составляющие исходную программу, в отдельные минимальные единицы текста, несущие смысловую нагрузку – лексемы.

Задача лексического анализа – перевести исходное текстовое представление из последовательности знаков в последовательность лексем. Эта последовательность символов затем передается на вход синтаксического анализатора.

Синтаксический анализ (парсинг, от англ parse – разбирать) производит построение общей структуры программы. Здесь имеет значение, какой символ

следует за текущим, а какой ему предшествует. Результат работы синтаксического анализатора имеет структура дерева – синтаксического дерева разбора.

Задача синтаксического анализатора (СиА) - провести разбор текста программы, сопоставив его с эталоном, данным в описании языка. Для синтаксического разбора используются контекстно-свободные грамматики (КС-грамматики).

Семантический анализ необходим для устранения особенностей языка, не поддающихся описанию средствами формальной грамматики. В частности, проверка типов и область видимости переменных происходит на семантической фазе анализа.

Выход семантического анализатора, таким образом, зачастую представляет собой последовательность инструкций, которая в функциональном смысле эквивалентна машинным инструкциям для некоей виртуальной машины.

Этапы синтаксического и семантического анализа обычно можно объединить.

## **2 ПОРЯДОК ВЫПОЛНЕНИЯ**

1. В соответствии с номером варианта составить описание модельного языка программирования в виде правил вывода формальной грамматики;
2. Составить таблицу лексем и нарисовать диаграмму состояний для распознавания и формирования лексем языка;
3. Разработать процедуру лексического анализа исходного текста программы на языке высокого уровня;
4. Разработать процедуру синтаксического анализа исходного текста методом рекурсивного спуска на языке высокого уровня;
5. Построить программный продукт, читающий текст программы, написанной на модельном языке, в виде консольного приложения;
6. Протестировать работу программного продукта с помощи серии тестов, демонстрирующих все основные особенности модельного языка программирования, включая возможные лексические и синтаксические ошибки.

### 3 ГРАММАТИКА МОДЕЛЬНОГО ЯЗЫКА

Согласно индивидуальному варианту задания на курсовую работу грамматика языка включает следующие синтаксические конструкции:

```
<операции_группы_отношения> ::= != | == | < | <= | > | >=
<операции_группы_сложения> ::= + | - | ||
<операции_группы_умножения> ::= * | / | &&
<унарная_операция> ::= !
<программа> = {/ (<описание> | <оператор>) ( : | переход
строки) /} end
<описание> ::= {<идентификатор> {, <идентификатор> } : <тип>
;}
<тип> ::= % | ! | $
<составной> ::= begin <оператор> { ; <оператор> } end
<присваивания> ::= <идентификатор> := <выражение>
<условный> ::= if «(>><выражение> <<)<<» <оператор> [else
<оператор>]
<фиксированного_цикла> ::= for <присваивания> to <выражение>
[step <выражение>] <оператор> next
<условного_цикла> ::= while «(>><выражение> <<)<<» <оператор>
<ввода> ::= readln идентификатор {, <идентификатор> }
<вывода> ::= writeln <выражение> {, <выражение> }
<Комментарии> ::= (* ... *)
<выражение> ::= <операнд> {<операции_группы_отношения>
<операнд>}
<операнд> ::= <слагаемое> {<операции_группы_сложения>
<слагаемое>}
```



```

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

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

<унарная_операция> <множитель> | «(» <выражение> «)»

<число> ::= <целое> | <действительное>

<логическая_константа> ::= true | false

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

    <буква> ::= A-Z | a-z

    <цифра> ::= 0-9

    <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 | d | e | f } (H | h)

```

Правила, описывающие действительные числа:

```

<действительное> ::= <числовая_строка> <порядок> |
[<числовая_строка>] . <числовая_строка> [порядок]

<числовая_строка> ::= { / <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. Жирным выделены терминалы, представляющие собой ключевые слова языка.

## 4 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 ограничители 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).

Исходные код лексического анализатора приведен в Приложении А.

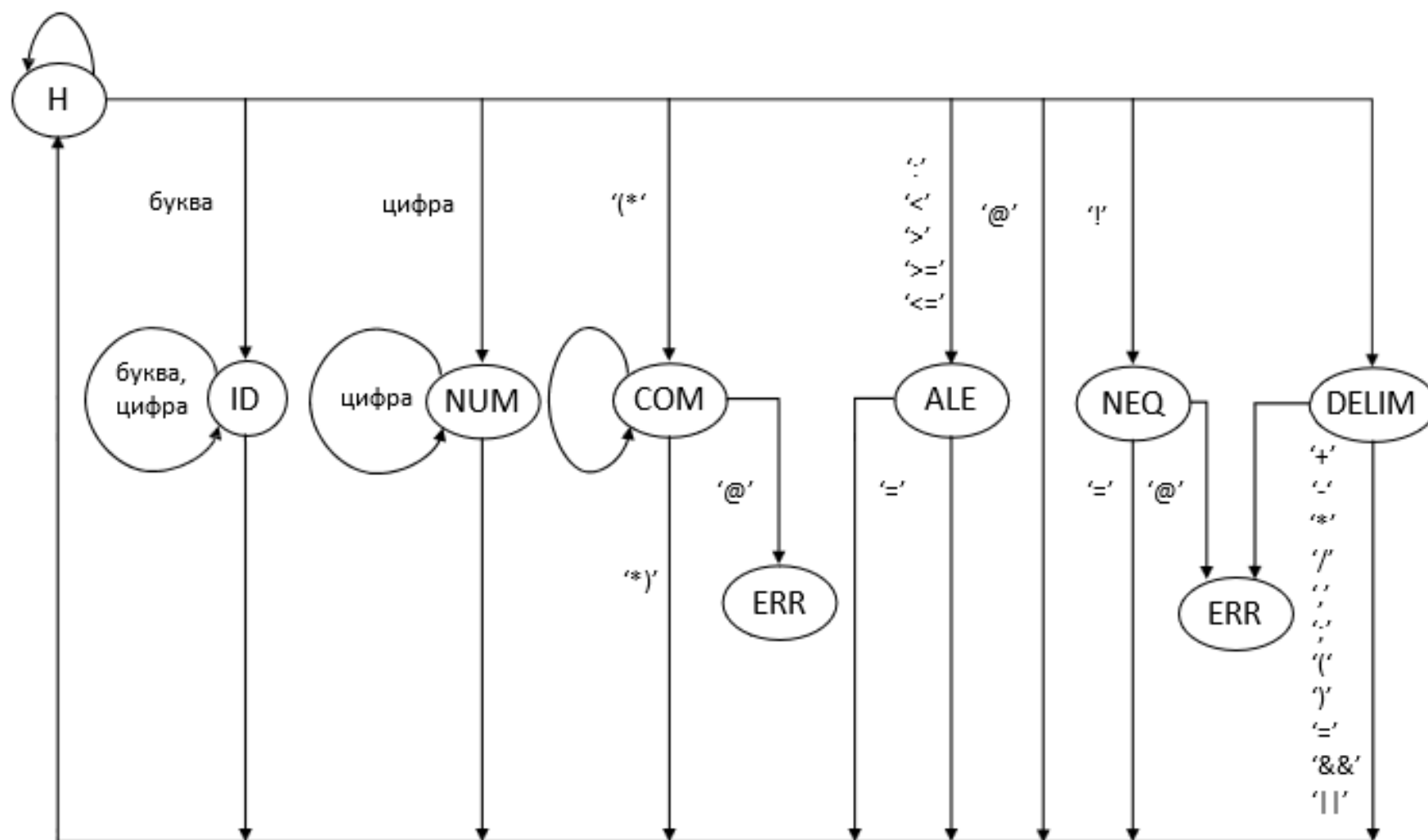


Рисунок 1 – диаграмма состояний лексического анализатора

## 5 ТЕСТИРОВАНИЕ ПРОГРАММЫ

В качестве программного продукта разработано консольное приложение *lexer.py*. Для тестирования программы был создан текстовый документ *input.txt*, содержащий набор ключевых слов, операторов, идентификаторов и чисел. Консольное приложение принимает на вход данные текстового файла и выдаёт в качестве результата документ *output.txt* в котором каждое слово (или же набор символов) с подписанными значениями после лексического анализа.

Исходный код программы для обработки лексическим анализатором приведен в листинге 1

*Листинг 1 – Содержание input.txt файла*

```
if a + b == c next;
for k to b;
!
%
&&
n != x
    step d begin 72 >= 34;
    (*commentaries*)
    writeln;

end;
```

*Листинг 2 – Содержание output.txt файла*

```
('IF', 'if', 1, 0)
('INDENT', 'a', 1, 3)
('PLUS', '+', 1, 5)
('INDENT', 'b', 1, 7)
('EQ', '==', 1, 9)
('INDENT', 'c', 1, 12)
('NEXT', 'next', 1, 14)
('DELIM', ';', 1, 18)
('FOR', 'for', 2, 0)
('INDENT', 'k', 2, 4)
('TO', 'to', 2, 6)
('INDENT', 'b', 2, 9)
('DELIM', ';', 2, 10)
```

('NOT', '!', 3, 0)  
('OST', '%', 4, 0)  
('AND', '&&', 5, 0)  
('INDENT', 'n', 6, 0)  
('NEQ', '!=', 6, 2)  
('INDENT', 'x', 6, 5)  
('STEP', 'step', 7, 4)  
('INDENT', 'd', 7, 9)  
('BEGIN', 'begin', 7, 11)  
('NUMBER', 72, 7, 17)  
('LE', '>=', 7, 20)  
('NUMBER', 34, 7, 23)  
('DELIM', ';', 7, 25)  
('COMMENT', '(\*commentaries\*)', 8, 4)  
('WRITELN', 'writeln', 9, 4)  
('DELIM', ';', 9, 11)  
('END', 'end', 11, 0)  
('DELIM', ';', 11, 3)

## ЗАКЛЮЧЕНИЕ

В работе представлен результат работы разработки лексического анализатора для модельного языка программирования. Грамматика языка задана с помощью формы Бэкуса-Наура (БНФ). Согласно грамматике, присутствуют лексемы следующих базовых типов: числовые константы, переменные, разделители и ключевые слова.

Лексический анализатор разработан на языке Python в виде функции Lex. Данная функция позволяет разделить последовательность символов исходного текста программы на последовательность лексем.

Тестирование программного продукта показало, что лексический анализ прошёл успешно и присутствует возможность разработки дальнейшего продукта с синтаксическими и сематическими функциями.

В ходе работы изучены основные принципы построения интеллектуальных систем на основе теории автоматов и формальных грамматик, приобретены навыки лексического анализа предложений языков программирования.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Унгер А.Ю. Теория формальных языков: учебное пособие. – Москва, 2024.
2. Документация встроенных фреймворков Python: инструментарий. – Нидерланды, 2024.



# **ПРИЛОЖЕНИЯ**

Приложение А – Класс лексического анализатора

## Приложение А

### Класс лексического анализатора

*Листинг А.1 – lexer.py*

```
import re
KEYWORDS = {
    'begin': 'BEGIN',
    'if': 'IF',
    'end': 'END',
    'for': 'FOR',
    'while': 'WHILE',
    'next': 'NEXT',
    'readln': 'READLN',
    'writeln': 'WRITELN',
    'to': 'TO',
    'step': 'STEP'
}

token_specification = [
    # Операторы:
    ('NEQ', r'!='),
    ('EQ', r'=='),
    ('LE', r'≤'),
    ('GE', r'≥'),
    ('LT', r'<'),
    ('GT', r'>'),
    ('PLUS', r'\+'),
    ('MINUS', r'-'),
    ('OR', r'\|\|'),
    ('MUL', r'\*'),
    ('DIV', r'/'),
    ('AND', r'&&'),
    ('NOT', r'!'),

    # Литералы:
    ('NUMBER', r'\d+(\.\d+)?'),
    ('INDENT', r'[A-Za-z_][A-Za-z0-9_]*'),

    ('NEWLINE', r'\n'),
    ('SKIP', r'[ \t\r]+'),

    ('MISMATCH', r'.')
]
```

### *Продолжение листинга A.1*

```
token_re = re.compile('|'.join('(?P<%s>%s)' % pair for pair in
token_specification))

def tokenize(code):
    line_num = 1
    line_start = 0
    for mo in token_re.finditer(code):
        kind = mo.lastgroup()
        value = mo.group()

        if kind == 'NUMBER':
            value = float(value) if '.' in value else int(value)

        elif kind == 'INDENT':
            if value in KEYWORDS:
                kind = KEYWORDS[value]

        elif kind == 'NEWLINE':
            # Подсчёт строк
            line_start = mo.end()
            line_num += 1
            continue

        elif kind == 'SKIP':
            continue

        elif kind == 'MISMATCH':
            raise RuntimeError(f'Неожиданный символ {value!r} на строке
{line_num}')
        yield kind, value, line_num, mo.start() - line_start

f = open("input.txt").readlines()
code = ""
for i in range(len(f)):
    code+=str(f[i])
# print(code)
fe = open("output_lexer.txt", "w")
try:
    for token in tokenize(code):
        print(token)
        fe.write(str(token) + '\n')
except ValueError as e:
    print(f"Error: {e}")
```