«Московский энергетический институт»  
Кафедра ПМИИ

**Отчёт по лабораторной работе №2**

**по курсу**

**«Основы построения трансляторов».**

**Тема: «Грамматический разбор методом рекурсивного спуска».**

**Выполнил:** Зиядов Э.А.

**Группа:** А-13-18.

**Вариант:** 8.

**Преподаватель:** Чернецов А.М.

Москва, 2021

**Задание.** Разработать алгоритм и реализовать программу для грамматического анализа методом рекурсивного спуска.

Для каждого индивидуального варианта задаётся набор операторов (подмножество языка Паскаль).

**Часть 1.** Необходимо описать правила языка в форме БНФ. По данным правилам описать грамматику языка. Разработанную грамматику преобразовать к форме автоматной грамматики. Результаты показать преподавателю.

**Часть 2.** По заданной грамматике построить ДКА, распознающий грамматику, и только её. Результаты показать преподавателю.

**Часть 3.** Разработать алгоритм синтаксического анализа методом рекурсивного спуска

Предусмотреть тесты для проверки распознавания всех лексем.

В каждом варианте должен быть реализован следующий функционал:

1. оператор присваивания;
2. арифметические операции + | - | \* | /;
3. арифметические выражения;
4. Типы данных: Integer, Boolean.

**Вариант 8.**

Program, var, const, begin, end, write, read, if, for, указатели.

----------------------------------------------------------------------------------------------------

**Часть 1.**

**Грамматика языка для индивидуального варианта в форме БНФ.**

1. <program> ::= <start\_prog> <main> | <main>
2. <start\_prog> ::= <program\_title> | <variables> | <program\_title> <variables>
3. <program\_title> ::= **program** <prog\_name>;
4. <prog\_name> ::= <id>
5. <id> ::= <letter> | <id> <letter> | <id> <number>
6. <letter> ::= **a | b | c | … | z | A | B | C | … | Z | \_**
7. <number> ::= **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |** <sign> <number>
8. <variables> ::= <def\_const> <def\_variables> | <def\_const> | <def\_variables> | <def\_variables> <def\_const>
9. <def\_const> ::= <const> | <def\_const>; <const>
10. <const> ::= **const** <c>;
11. <c> ::= <id> **=** <constant> | <c>; <id> **=** <constant>
12. <constant> ::= <math\_expr>
13. <math\_expr> ::= <term> | <math\_expr> + <term> | <math\_expr> - <term>
14. <term> ::= <factor> | <term> \* <factor> | <term> **div** <factor>
15. <factor> ::= <id> | <id>^ | <integer>
16. <integer> ::= <id> | <number> | <sign> <number> | <integer> <number>
17. <sign> ::= **-** | **+**
18. <def\_variables> ::= <var> | <def\_variables>; <var>
19. <var> ::= **var** <v>;
20. <v> ::= <id\_list>: <type>
21. <id\_list> ::= <id> | <id\_list>, <id>
22. <type> ::= **integer** | **boolean** | ^**integer** | ^**boolean**
23. <main> ::= **begin end.** | **begin** <code> **end.**
24. <code> ::= <stmt> | <code>; <stmt> | **begin end | begin** <code> **end**
25. <stmt> ::= <assign> | <read> | <write> | <if> | <for>
26. <assign> ::= <id> := <expr>; | <id> := **@**<id>; | <id> := **nil**; | <id> := <id>^;
27. <read> ::= **read**(<id\_list>); | **read();**
28. <write> ::= **write**(<id\_list>); | **write();**
29. <for> ::= **for** <index\_expr> **do** <body>
30. <index\_expr> ::= <id> **:=** <math\_expr> **to** <math\_expr> | <id> **:=** <math\_expr> **downto** <math\_expr>
31. <body> ::= <stmt> | **begin** <code> **end;**
32. <if> ::= **if** <logic\_expr> **then** <body> | **if** <logic\_expr> **then** <body> **else** <body>
33. <logic\_expr> ::= <simple\_logic\_expr> | **not** <simple\_logic\_expr> | <simple\_logic\_expr> <logic\_operator> <simple\_logic\_expr> | <math\_expr> <relational\_operators> <math\_expr>
34. <logic\_operator> ::= **=** | **<>** | **<** | **<=** | **>=** | **>** | **or** | **and** | **xor**
35. <simple\_logic\_expr> ::= <id> | <id>^ | <boolean> | <integer> <relational\_operators> <integer> | <boolean> <logic\_operator> <boolean>
36. <relational\_operators> ::= **=** | **<>** | **<** | **<=** | **>=** | **>**
37. <boolean> ::= <id> | **true** | **false**

**Множество терминальных символов T** = { begin, end, program, const, var, integer, boolean, ^integer, ^boolean, pointer, nil, @, ^, read, write, for, do, to, if, then, else, \_, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \*, div, not, or, and, xor, true, false, “.”, “,”, “;”, :=, =, >, < }.

**Множество нетерминальных символов N** = { <write>, <type>, <expr>, <program>, <number>, <integer>, <for>, <program\_title>, <var>, <stmt>, <code>, <factor>, <relational\_operators>, <variables>, <body>, <read>, <logic\_operator>, <assign>, <index\_expr>, <id\_list>, <if>, <def\_variables>, <def\_const>, <simple\_logic\_expr>, <sign>, <boolean>, <v>, <letter>, <const>, <empty>, <math\_expr>, <logic\_expr>, <c>, <constant>, <term>, <prog\_name>, <id> }.

**Начальный символ Z** = { <program> }.

**Автоматная грамматика:**

S – начальный символ.

S → program S1 | const S2 | var S3

// Имя программы  
S1 → A-Z S4 | a-z S4 | \_ S4  
S4 → A-Z S4 | a-z S4 | \_ S4 | 0-9 S4 | ; S5  
S5 → const S2 | var S3

// Объявление переменных  
S3 → A-Z S11 | a-z S11 | \_ S11  
S11 → A-Z S11 | a-z S11 | \_ S11 | 0-9 S11 | , S3 | : S12  
S12 → integer S13 | boolean S13 | ^integer S13 | ^boolean S13  
S13 → ; S9

// Объявление констант  
S2 → A-Z S6 | a-z S6 | \_ S6  
S6 → A-Z S6 | a-z S6 | \_ S6 | 0-9 S6 | = S7  
S7 → <expr> S8  
S8 → ; S9  
S9 → const S2 | var S3 | begin S10

// Основная программа  
S10 → if S14 | for S15 | read S16 | write S17 | A-Z S18 | a-z S18 | \_ S18 | begin S37 | end. S\_last

// Вложенность  
S37 → if S38 | for S39 | read S40 | write S41 | A-Z S42 | a-z S42 | \_ S42 | end; S10  
…

// Условный оператор  
S14 → <logic\_expr> S19  
S19 → then S20  
S20 → <stmt> S21 | begin S22  
S21 → ; S10  
S22 → end; S10 | <stmt> S23  
S23 → ; S24  
S24 → <stmt> S23 | end; S10  
  
// Цикл for  
S15 → A-Z S19 | a-z S25  
S25 → A-Z S25 | a-z S25 | \_ S25 | 0-9 S25 | := S26  
S26 → <math\_expr> S27  
S27 → to S28 | downto S28  
S28 → <math\_expr> S29  
S29 → do S20  
  
// read  
S16 → ( S30  
S30 → ) S31 | A-Z S32 | a-z S32 | \_ S32  
S31 → ; S10  
S32 → a-z S32 | A-Z S32 | 0-9 S32 | \_ S32 | , S33 | ) S34  
S33 → A-Z S32 | a-z S32 | \_ S32  
S34 → ; S10

// write  
S17 → ( S30

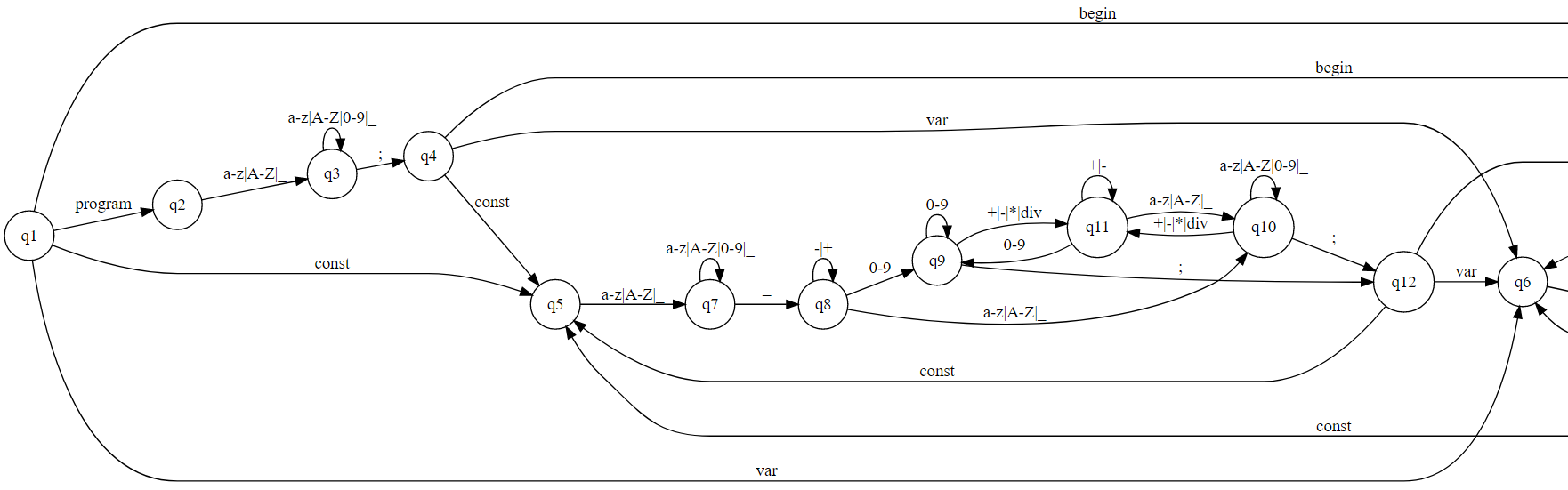
// Присваивание  
S18 → a-z S18 | A-Z S18 | 0-9 S18 | \_ S18 | := S35  
S35 → <math\_expr> S36  
S36 → ; S10

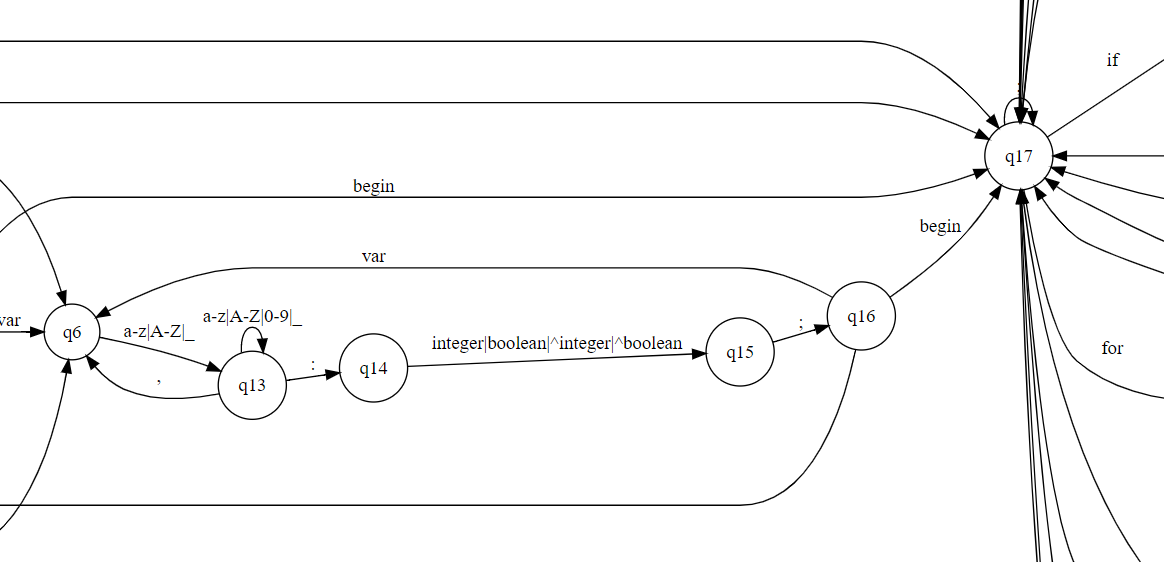
**Часть 2.**

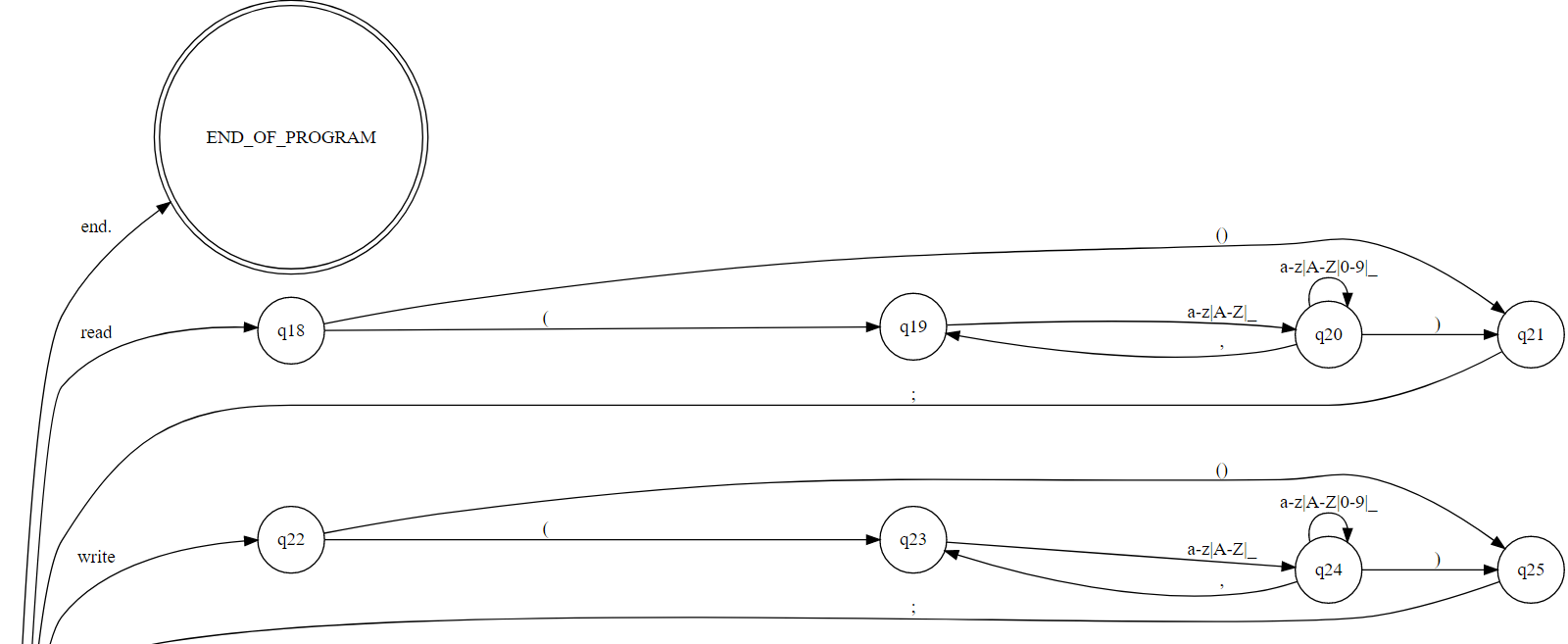
**ДКА.**

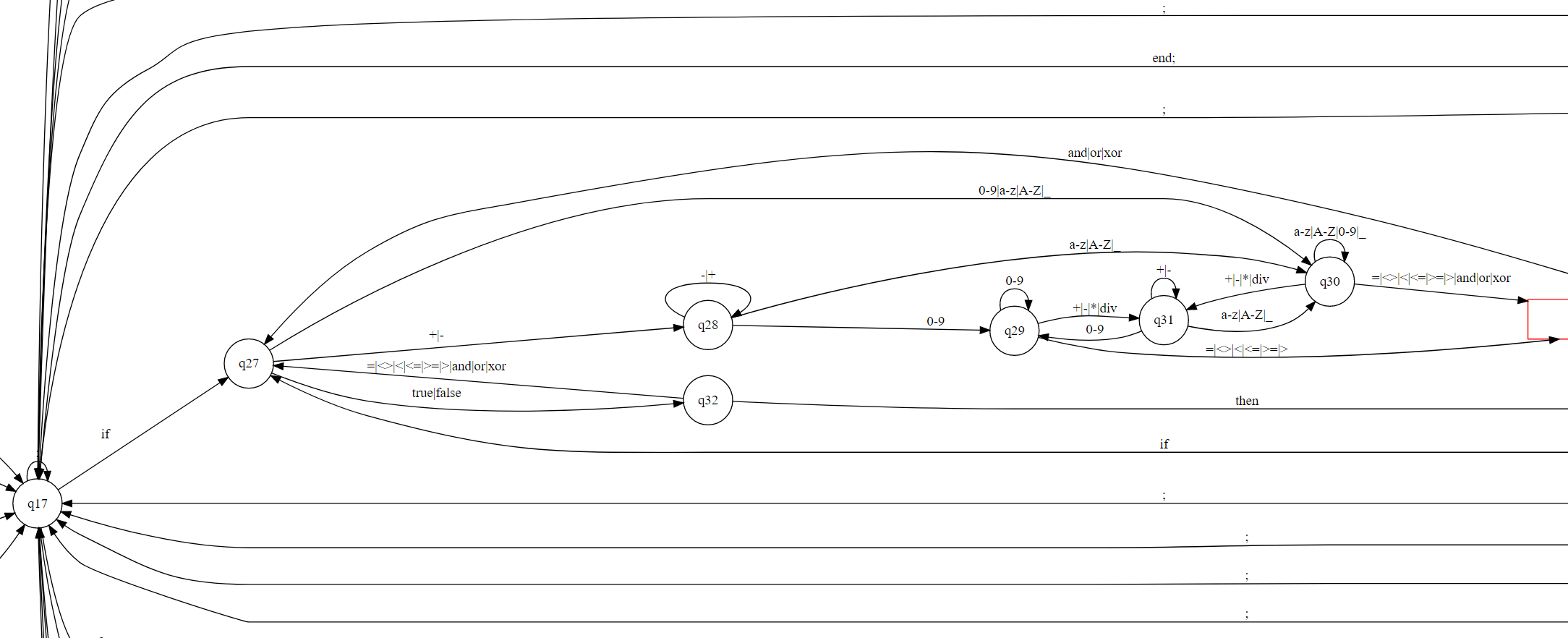
Ниже приведены части одного большого ДКА, так как полностью изображение автомата не вместилось на страницу.

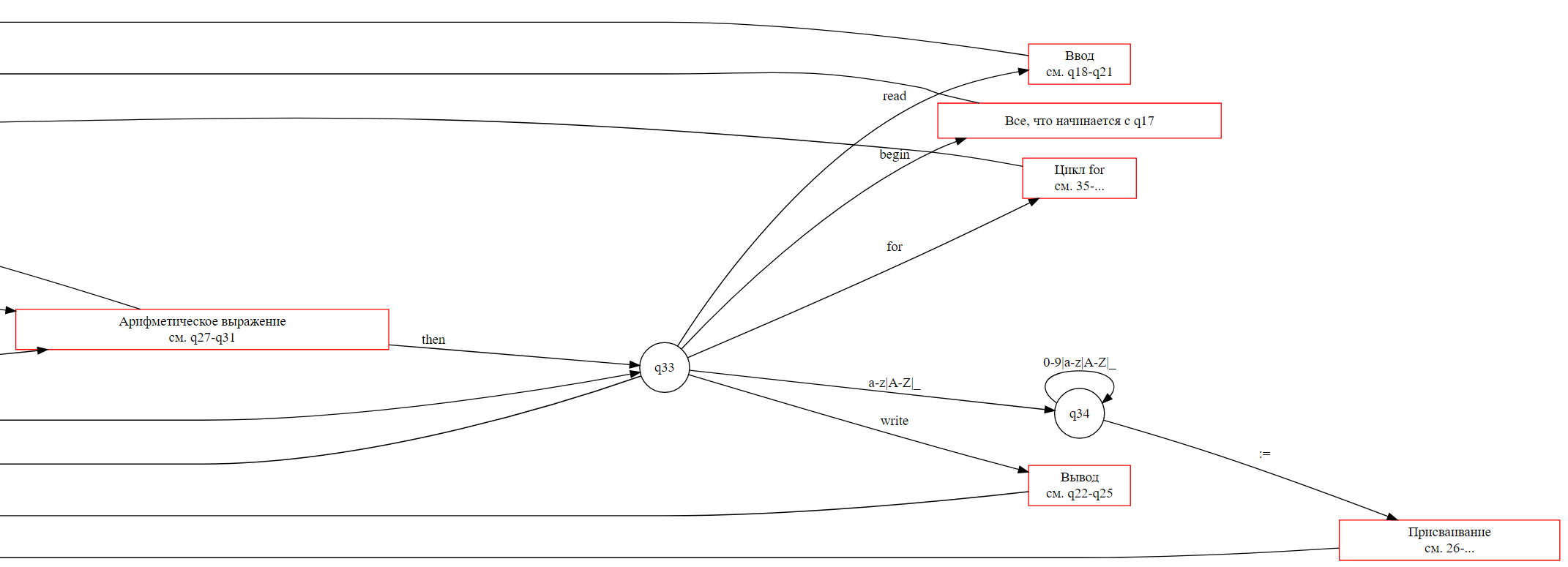
Начало программы:



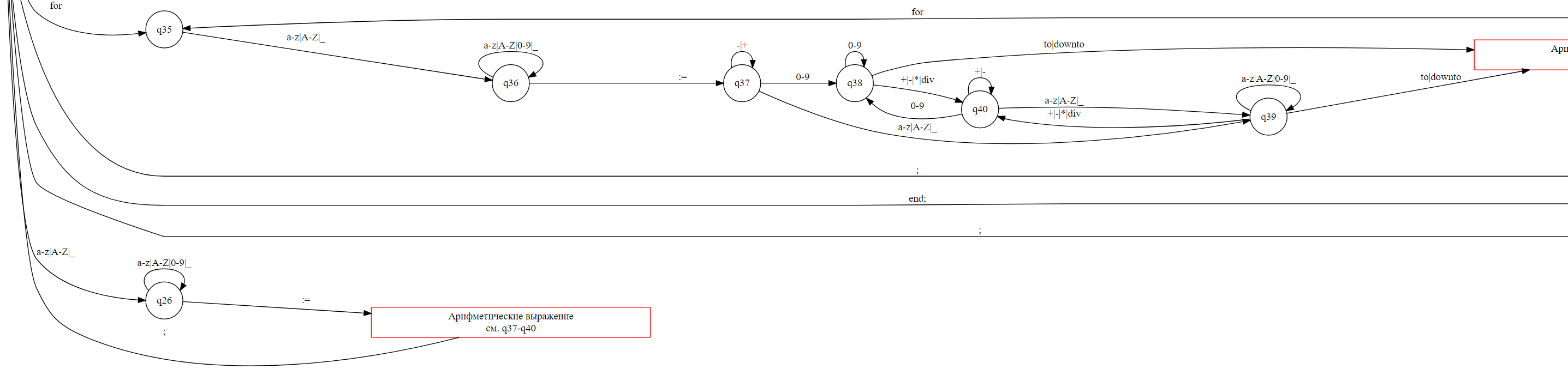


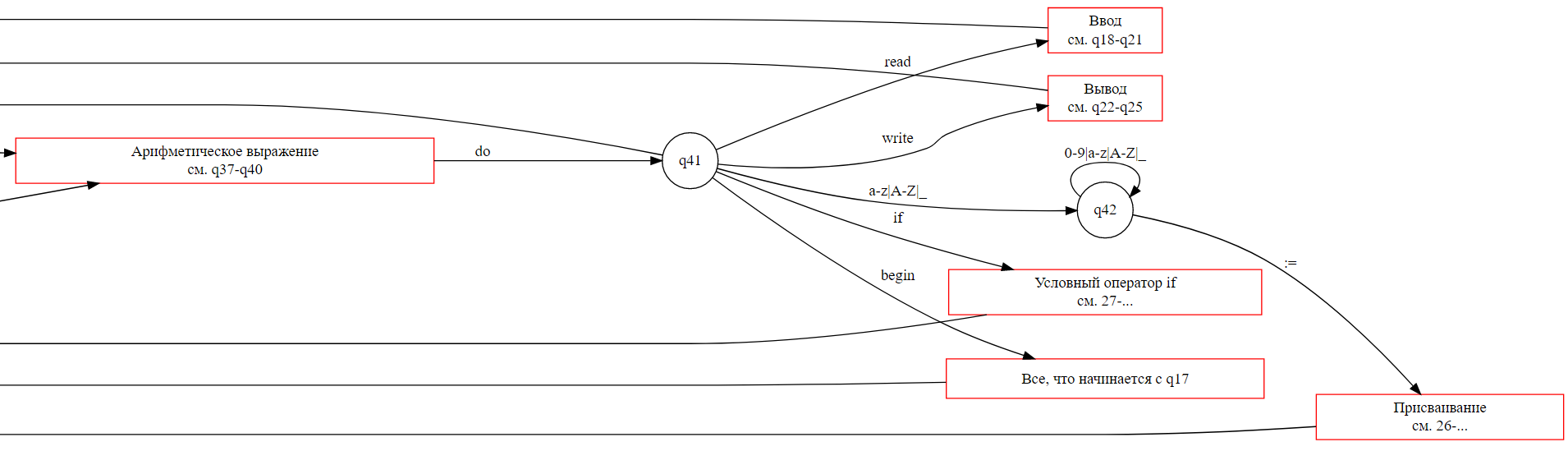
Терминальное состояние, read и write.  


Блок if.  




Блок for и присваивание.





**Часть 3.**

**Анализ методом рекурсивного спуска.**

**Метод рекурсивного спуска** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Recursive descent parser*) — алгоритм [нисходящего синтаксического анализа](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D1%8F%D1%89%D0%B8%D0%B9_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7), реализуемый путём [взаимного вызова](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) процедур, где каждая процедура соответствует одному из правил [контекстно-свободной грамматики](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BD%D0%BE-%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) или [БНФ](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0_%D0%91%D1%8D%D0%BA%D1%83%D1%81%D0%B0_%E2%80%94_%D0%9D%D0%B0%D1%83%D1%80%D0%B0). Применения правил последовательно, слева-направо поглощают [токены](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BA%D0%B5%D0%BD_(%D0%BB%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7)), полученные от [лексического анализатора](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7). Это один из самых простых алгоритмов синтаксического анализа, подходящий для полностью ручной реализации.

**Алгоритм:**

∙ Первоначально на вход анализатору поступает строка, которая представляет собой текст программы.  
Затем эта строка приводится к нижнему регистру (так как язык Паскаль не учитывает регистр в программном коде) и разбивается на токены (программно это массив строк).

∙ Массив токенов поступает на вход главной функции, которая затем последовательно вызывает остальные функции (реализуется рекурсивный спуск).  
Каждая функция соответсвует нетерминалу из БНФ.  
Функция выбрасывает исключение при некорректном токене, иначе передает управление следующей функции.  
Таким образом, если за все время работы анализатора не было исключений – код корректен.

∙ Сначала вызывается функция, которая отвечает за декларативную часть программы.  
Если встретилось слово ‘program’, то затем проверяется корректность идентификатора и наличие после него точки с запятой.  
Затем проверяется наличие блока ‘const’ и/или ‘var’ (они могут быть объявлены в любом порядке).  
В поле констант проверяются корректность идентификаторов и выражений (между ними должен быть знак “=”) в поле переменных корректность идентификаторов и типов (между ними должен быть знак “:”).   
Следующим шагом проверяется наличие главного “begin” для начала программы. При отсутствии выбрасывается исключенение.  
Внутри основного блока рассматривается любая последовательность операторов (if .. else, for, присваивание, read, write), для каждого из них есть функция, которая проверяет корректность записи (для read и write – сначала должна идти открывающая скобка, затем либо пустота, либо список идентификаторов строго через запятую, в конце – закрывающая скобка).   
После каждого оператора проверяется наличие точки с запятой (обрабатывается случай, что перед else точка с запятой не ставится).  
Внутри if и for при наличи операторных слов begin … end; может быть также любая последовательность операторов, иначе – лишь один оператор.   
Если все операторы обработаны и они корректны – проверяется наличие “end” для главного “begin”. Если его нет – ошибка. Если нет точки после него – ошибка.  
Если после “end.” в программе есть что-то еще – ошибка.

Если исключений не возникло и алгоритм прекратил работу – успех. Программа корректна и соответсвует написанной БНФ.

**Тесты:**

1. Минимально корректная программа.

begin  
end.

**Результат – correct!**

2. Корректная программа с блоками var и const без операторов.

program abc;

const a = 2 + 2;  
var a: integer;

begin  
end.

**Результат – correct!**

3. Ошибка в идентификаторе.

program 1abc;

begin  
end.

**Результат – Incorrect identitfacator on line 1.**

4. Ошибка в блоке var.

var a = ^integer;

begin  
end.

**Результат – Found ‘=’ but expected ‘:’ on line 1.**

5. Ошибка в блоке const.

const a : (2 \* 2) div 5;

begin  
end.

**Результат – Found ‘:’ but expected ‘=’ on line 1.**

6. Ошибка при присваивании.

begin  
 a = b^;  
end.

**Результат – Found ‘=’ but expected ‘:=’ on line 2.**

7. Ошибка в блоке for.

begin  
 for i := 1 to 5 read();  
end.

**Результат – Found ‘read’ but expected ‘do’ on line 2.**

8. Ошибка в read.

begin  
 read()  
end.

**Результат – Found ‘end’ but expected ‘;’ on line 3.**

9. Ошибка в if.

begin  
 if a xor b and (3 > 2) write();  
end.

**Результат – Found ‘write’ but expected ‘then’ on line 2.**

10. else без if.

begin  
 else write(a, b, c);  
end.

**Результат – Undefined operator on line 2.**

11. Нет точки после главного end.

begin  
end  
**Результат – Found end of file but expected ‘.’ on line 2.**

12. Пустая программа.

**Результат – Found end of file but declaration section or begin expected on line 1.**

13. Корректная программа.

program abc;

const a = (-2 \* 2) div 5 + (3 - +3) \* -5;

var i, a, b: integer;

flag: boolean;

begin

b := (2 \* 3 - 5 div (5 + 5));

a := @b;

b := a^ + a^;

for i := 1 to 100 + 1 do

flag := false;

begin

flag := true;

for i := 100 + 1 downto 1 do

if (i < -5) and (not flag) then

begin

read(a);

write(a);

end

else

begin

read(a, b, c);

for a := 15 to a do

begin

write(a, b, c, d);

end;

write();

read();

end;

end;

if true then

read()

else

write();

end.

**Результат – correct!**

**Программный код (на языке Python):**

Файл Lab2.py (интерфейс):

import tkinter as tk

from tkinter import ttk

import ctypes

import sys

from tkinter.constants import END

import tkinter.messagebox as mb

from analyzer import Analyzer

############################################################# Создание интерфейса ############################################################

# Для высокого разрешения

ctypes.windll.shcore.SetProcessDpiAwareness(1)

root = tk.Tk()

root.title('Pascal subset')

root.geometry('1000x800+450+100')

error\_line = 0

# Перенос строк отключен, иначе нумерация будет работать некорректно

numbers = tk.Text(root, width=4, bg='lightgray', state=tk.DISABLED, relief=tk.FLAT)

numbers.grid(row=0, column=0, sticky='NS')

scroll = ttk.Scrollbar(root)

scroll.grid(row=0, column=2, sticky='NS')

def on\_yscrollcommand(\*args):

    scroll.set(\*args)             # Синхронизация скролбара с текстовым полем

    numbers.yview\_moveto(args[0]) # Синхронизация поля с номерами с текстовым полем

text = tk.Text(root, yscrollcommand=on\_yscrollcommand, wrap=tk.NONE)

text.grid(row=0, column=1, sticky='NSWE')

def scroll\_command(\*args):

    # Движение скролбара управляет отображением текста в обоих текстовых полях

    text.yview(\*args)

    numbers.yview(\*args)

scroll.config(command=scroll\_command)

def insert\_numbers():

    count\_of\_lines = text.get(1.0, tk.END).count('\n') + 1

    numbers.config(state=tk.NORMAL)

    numbers.delete(1.0, tk.END)

    numbers.insert(1.0, '\n'.join(map(str, range(1, count\_of\_lines))))

    numbers.config(state=tk.DISABLED)

insert\_numbers()

def on\_edit(event):

    # Срабатывает при изменениях в текстовом поле

    global error\_line

    text.tag\_delete('failed')

    insert\_numbers()

    text.edit\_modified(0) # Сбрасываем флаг изменения текстового поля

text.bind('<<Modified>>', on\_edit)

def tab(arg):

    text.insert(tk.INSERT, " " \* 4)

    return 'break'

text.bind('<Tab>', tab)

# Нужно, чтобы текстовое поле автоматически меняло размер при изменении размера окна

root.grid\_columnconfigure(1, weight=1)

root.grid\_rowconfigure(0, weight=1)

#################################################################### МЕНЮ ##################################################################

mainmenu = tk.Menu(root)

root.config(menu=mainmenu)

filemenu = tk.Menu(mainmenu, tearoff=0)

################################################################## Кнопка анализа ##########################################################

def analyze():

    global error\_line

    s = text.get(1.0, END)

    analyzer = Analyzer(s)

    # Следующий вывод для теста, он ни на что не влияет.

    #print(analyzer.program1)

    #print(analyzer.program2)

    try:

        analyzer.check()

        mb.showinfo(title='Congratulations!', message='Correct!')

    except Exception as ex:

        number\_error\_line = analyzer.error\_line

        text.tag\_add('failed', number\_error\_line + 0.0, number\_error\_line + 0.777)

        text.tag\_config('failed', background='red')

        mb.showerror(title="Error info", message=ex)

filemenu.add\_command(label="Analysis", command=analyze)

filemenu.add\_separator()

############################################################################################################################################

def Close():

    root.destroy()

filemenu.add\_command(label="Exit", command=Close)

helpmenu = tk.Menu(mainmenu, tearoff=0)

def about\_message():

    mb.showinfo(

        title="About \'Pascal subset\'",

        message="Лабораторная работа 2 по ОПТ.\nРеализация подмножества языка Pascal.\nНИУ МЭИ, Зиядов Э.А., группа А-13-18, вариант 8."

    )

helpmenu.add\_command(label="About", command=about\_message)

mainmenu.add\_cascade(label="File", menu=filemenu)

mainmenu.add\_cascade(label="Help", menu=helpmenu)

############################################################# Конец создания интерфейса #########################################################

s = """program abc;\n\nconst a = (-2 \* 2) div 5 + (3 - +3) \* -5;\nvar i, a, b: integer; \n    flag: boolean;\n\nbegin\n    \n    b := (2 \* 3 - 5 div (5 + 5));\n    a := @b;\n    b := a^ + a^;\n        \n

    for i := 1 to 100 + 1 do\n    flag := false;\n    begin\n        flag := true;\n        for i := 100 + 1 downto 1 do\n        if (i < -5) and (not flag) then\n        begin\n            read(a);\n            write(a);\n        end\n        else\n        begin\n            read(a, b, c);\n            for a := 15 to a do\n            begin\n                write(a, b, c, d);\n            end;\n            write();\n            read();\n        end;\n    end;\n\n    if true then \n        read()\n    else\n        write();\n\nend.\n"""

text.insert(1.0, s)

mb.showinfo(title='Информация.', message='В окне приведен пример корректной программы.\nДля ее анализа надо нажать в меню "файл -> анализ".\nВы можете ввести в текстовое\

 поле любой текст,\nа далее проанализировать его - является ли он корректным с точки зрения языка Паскаль.\n(Редактирование доступно только после одного нажатия на кнопку "анализ")')

if \_\_name\_\_ == '\_\_main\_\_':

    root.mainloop()

Файл analyzer.py (класс анализатора):

from my\_parser import Parser

import re

# Класс для ошибок

class Error(Exception):

    def \_\_init\_\_(self, msg, pos, \*args):

        self.pos = pos

        self.msg = msg

        self.args = args

    def \_\_str\_\_(self):

        return f'{self.msg} on line {self.pos}.'

# Класс анализатора

class Analyzer:

    '''

    1.  <program> ::= <start\_prog> <main> | <main>

    2.  <start\_prog> ::= <program\_title> | <variables> | <program\_title> <variables>

    3.  <program\_title> ::= program <prog\_name>;

    4.  <prog\_name> ::= <id>

    5.  <id> ::= <letter> | <id> <letter> | <id> <number>

    6.  <letter> ::= a | b | c | … | z | A | B | C | … | Z | \_

    7.  <number> ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | <sign> <number>

    8.  <variables> ::= <def\_const> <def\_variables> | <def\_const> | <def\_variables> | <def\_variables> <def\_const>

    9.  <def\_const> ::= <const> | <def\_const>; <const>

    10. <const> ::= const <c>;

    11. <c> ::= <id> = <constant> | <c>; <id> = <constant>

    12.  <constant> ::= <math\_expr>

    13.  <math\_expr> ::= <term> | <math\_expr> + <term> | <math\_expr> - <term>

    14.  <term> ::= <factor> | <term> \* <factor> | <term> div <factor>

    15.  <factor> ::= <id> | <id>^ | <integer>

    16.  <integer> ::= <id> | <number> | <sign> <number> | <integer> <number>

    17.  <sign> ::= - | +

    18.  <def\_variables> ::= <var> | <def\_variables>; <var>

    19.  <var> ::= var <v>;

    20.  <v> ::= <id\_list>: <type> ; <v>; <id\_list>: <type>

    21.  <id\_list> ::= <id> | <id\_list>, <id>

    22.  <type> ::= integer | boolean | ^integer | ^boolean

    23.  <main> ::= begin end. | begin <code> end.

    24.  <code> ::= <stmt> | <code>; <stmt> | begin end | begin <code> end

    25.  <stmt> ::= <assign> | <read> | <write> | <if> | <for>

    26.  <assign> ::= <id> := <expr>; | <id> := @<id>; | <id> := nil; | <id> := <id>^;

    27.  <read> ::= read(<id\_list>); | read();

    28.  <write> ::= write(<id\_list>); | write();

    29.  <for> ::= for <index\_expr> do <body>

    30.  <index\_expr> ::= <id> := <math\_expr> to <math\_expr> | <id> := <math\_expr> downto <math\_expr>

    31.  <body> ::= <stmt> | begin  <code> end;

    32.  <if> ::= if <logic\_expr> then <body> | if <logic\_expr> then <body> else <body>

    33. <logic\_expr> ::= <simple\_logic\_expr> | not <simple\_logic\_expr> | <simple\_logic\_expr> <logic\_operator> <simple\_logic\_expr>  | <math\_expr> <relational\_operators> <math\_expr>

    34. <logic\_operator> ::= = | <> | < | <= | >= | > | or | and | xor

    35.  <simple\_logic\_expr> ::= <id> | <id>^ | <boolean> | <integer> <relational\_operators> <integer> | <boolean> <logic\_operator> <boolean>

    36.  <relational\_operators> ::= = | <> | < | <= | >= | >

    37.  <boolean> ::= <id> | true | false

    '''

    # program\_text - текст исходной программы

    def \_\_init\_\_(self, program\_text):

        self.program\_text = program\_text

        parser = Parser(program\_text)

        self.program1 = parser.to\_string\_list1()

        self.program2 = parser.to\_string\_list2()

        self.cur\_ind = 0

        self.error\_ind = 0

        self.error\_line = 0

        self.counter = 0 # счетчик скобок

        self.flag = False

    # Функция, которая выбрасывает ошибку и определяет строчку с ошибкой

    def \_\_throw\_error(self, cur\_ind, msg, \*args):

        self.error\_ind = cur\_ind

        self.error\_line = self.get\_number\_error\_line()

        raise Error(msg, self.error\_line, \*args)

    # Выдает строку, на которой нашлась ошибка

    def get\_number\_error\_line(self):

        line\_number = 1

        ind = 0

        for word in self.program2:

            if word == '\n':

                line\_number += 1

            else:

                if ind == self.error\_ind:

                    return line\_number

                ind += 1

        return 1

    # <id> ::= <letter> | <id> <letter> | <id> <number>

    # <letter> ::= a | b | c | … | z | A | B | C | … | Z | \_

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

    def \_\_id(self, pointer=False, adr=False):

         if self.program1[self.cur\_ind] in ['begin', 'end', 'program',

                                            'const', 'var', 'integer',

                                            'boolean', '^integer', '^boolean',

                                            'pointer', 'nil', 'write',

                                            'for', 'do', 'to',

                                            'if', 'then', 'else', 'div', 'read']:

            return False

         if pointer and adr:

             pattern = re.compile('^([@]?[a-z\_])+[0-9a-z\_]\*[\^]?$')

         elif pointer:

             pattern = re.compile('^[a-z\_]+[0-9a-z\_]\*[\^]?$')

         elif adr:

             pattern = re.compile('^([@]?[a-z\_])+[0-9a-z\_]\*$')

         else:

             pattern = re.compile('^[a-z\_]+[0-9a-z\_]\*$')

         if pattern.match(self.program1[self.cur\_ind]) is not None:

             return True

         else:

            return False

    # <prog\_name> ::= <id>

    def \_\_prog\_name(self):

        return self.\_\_id()

    # <program\_title> ::= program <prog\_name>;

    def \_\_program\_title(self):

        if self.program1[self.cur\_ind] == 'program':

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but identificator expected', 120)

            if self.\_\_prog\_name():

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but \';\' expected', 126)

                if self.program1[self.cur\_ind] != ";":

                    self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but \';\' expected', 129)

            else:

                self.\_\_throw\_error(self.cur\_ind, 'Incorrect identificator', 132)

    # <integer> ::= <id> | <number> | <sign> <number> | <integer> <number>

    # <sign> ::= - | +

    def \_\_integer(self):

        if re.match(r'[+-]\*[0-9]+$', self.program1[self.cur\_ind]) is not None:

            return True

        else:

            return False

    # <factor> ::= <id> | <id>^ | <integer>

    def \_\_factor(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 146)

        self.\_\_check\_brackets()

        if not self.\_\_id(pointer=True) and not self.\_\_integer():

            self.\_\_throw\_error(self.cur\_ind, f'Incorrect expression', 149)

    # <term> ::= <factor> | <term> \* <factor> | <term> div <factor>

    # Избавимся от левой рекурсии:

    # <term> ::= <factor>[\*<factor>]\* | <factor>[div <factor>]\*

    def \_\_term(self):

        self.\_\_factor()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 159)

        self.\_\_check\_brackets()

        if self.program1[self.cur\_ind] in ['\*', 'div']:

            self.\_\_factor()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 166)

            self.\_\_check\_brackets()

            if self.program1[self.cur\_ind] in ['\*', 'div']:

                self.\_\_term()

            else:

                self.cur\_ind -= 1

        else:

            self.cur\_ind -= 1

    # Функция проверяет баланс скобок в выражении.

    def \_\_check\_brackets(self):

        if self.program1[self.cur\_ind] == '(':

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 185)

            self.counter += 1

            while self.program1[self.cur\_ind] == '(':

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 185)

                self.counter += 1

        elif self.program1[self.cur\_ind] == ')':

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 185)

            self.counter -= 1

            while self.program1[self.cur\_ind] == ')':

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 185)

                self.counter -= 1

    # <math\_expr> ::= <term> | <math\_expr> + <term> | <math\_expr> - <term>

    # Избавимся от левой рекурсии:

    # <math\_expr> ::= <term>[-<term>]\* | <term>[+<term>]\*

    def \_\_math\_expr(self, log = False):

        self.\_\_term()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 185)

        self.\_\_check\_brackets()

        if self.program1[self.cur\_ind] in ['+', '-']:

            self.\_\_term()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 192)

            self.\_\_check\_brackets()

            if self.program1[self.cur\_ind] in ['+', '-']:

                self.\_\_math\_expr()

            else:

                if self.counter != 0 and not log:

                    self.\_\_throw\_error(self.cur\_ind, f'Incorrect expression', 185)

                self.cur\_ind -= 1

        else:

            if self.counter != 0 and not log:

                self.\_\_throw\_error(self.cur\_ind, f'Incorrect expression', 185)

            self.cur\_ind -= 1

    # <constant> ::= <math\_expr>

    def \_\_constant(self):

        self.\_\_math\_expr()

    # <c> ::= <id> = <constant>; | <c>; <id> = <constant>;

    # Избавимся от левой рекурсии:

    # <c> ::= [<id> = <constant>;]\*

    def \_\_c(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected identificator', 213)

        if self.\_\_id():

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but \'=\' expected', 219)

            if self.program1[self.cur\_ind] != '=':

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but \'=\' expected', 221)

            self.\_\_constant()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but \';\' expected', 226)

            if self.program1[self.cur\_ind] != ';':

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but \';\' expected', 228)

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected identificator', 232)

            if self.\_\_id():

                self.cur\_ind -= 1

                self.\_\_c()

            else:

                if self.program1[self.cur\_ind] not in ['begin', 'const', 'var']:

                    self.\_\_throw\_error(self.cur\_ind, 'Incorrect identificator', 239)

                self.cur\_ind -= 1

        else:

            self.\_\_throw\_error(self.cur\_ind, 'Incorrect identificator', 243)

    # <const> ::= const <c>

    def \_\_const(self):

        if self.cur\_ind == 0 and self.program1[self.cur\_ind] == 'const':

            self.\_\_c()

        elif self.cur\_ind != 0:

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                if len(self.program1) == 1: return

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected begin', 252)

            if self.program1[self.cur\_ind] == 'const':

                self.\_\_c()

            else:

                self.cur\_ind -= 1

    # <def\_const> ::= <const> | <def\_const> <const>

    # Избавимся от левой рекурсии:

    # <def\_const> ::= <const> | <const> <def\_const>

    def \_\_def\_const(self):

        self.\_\_const()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            if len(self.program1) == 1:

                self.cur\_ind -= 1

                return

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected begin', 267)

        if self.program1[self.cur\_ind] == 'const':

            self.cur\_ind -= 1

            self.\_\_def\_const()

        elif self.program1[self.cur\_ind] == 'var':

            self.cur\_ind -= 1

            self.\_\_def\_variables()

        else:

            self.cur\_ind -= 1

    # <id\_list> ::= <id> | <id\_list>, <id>

    def \_\_id\_list(self, pointer=False, adr=False, int=False):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected identificator', 283)

        if self.\_\_id(pointer, adr) or (int and self.\_\_integer()):

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \':\'', 288)

            if self.program1[self.cur\_ind] == ',':

                self.\_\_id\_list(pointer, adr, int)

            else:

                self.cur\_ind -= 1

        else:

            self.\_\_throw\_error(self.cur\_ind , f'Incorrect identificator', 295)

    # <type> ::= integer | boolean | ^integer | ^boolean

    def \_\_type(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected type', 302)

        if self.program1[self.cur\_ind] not in ['integer', 'boolean', '^integer', 'boolean']:

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected type', 305)

    # <v> ::= <id\_list>: <type>; | <v>; <id\_list>: <type>;

    # Избавимся от левой рекурсии:

    # <v> ::= [<id\_list>: <type>;]\*

    def \_\_v(self):

        self.\_\_id\_list()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \':\'', 315)

        if self.program1[self.cur\_ind] != ':':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \':\'', 318)

        self.\_\_type()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but \';\' expected', 324)

        if self.program1[self.cur\_ind] != ';':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but \';\' expected', 326)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected identificator', 330)

        if self.\_\_id():

            self.cur\_ind -= 1

            self.\_\_v()

        else:

            if self.program1[self.cur\_ind] not in ['begin', 'const', 'var']:

                    self.\_\_throw\_error(self.cur\_ind, 'Incorrect identificator', 337)

            self.cur\_ind -= 1

    # <var> ::= var <v>

    def \_\_var(self):

        if self.cur\_ind == 0 and self.program1[self.cur\_ind] == 'var':

            self.\_\_v()

        elif self.cur\_ind != 0:

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                if len(self.program1) == 1: return

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected begin', 347)

            if self.program1[self.cur\_ind] == 'var':

                self.\_\_v()

            else:

                self.cur\_ind -= 1

    # <def\_variables> ::= <var> | <def\_variables> <var>

    # Избавимся от левой рекурсии:

    # <def\_variables> ::= <var> | <var> <def\_variables>

    def \_\_def\_variables(self):

        self.\_\_var()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            if len(self.program1) == 1:

                self.cur\_ind -= 1

                return

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected begin', 362)

        if self.program1[self.cur\_ind] == 'var':

            self.cur\_ind -= 1

            self.\_\_def\_variables()

        elif self.program1[self.cur\_ind] == 'const':

            self.cur\_ind -= 1

            self.\_\_def\_const()

        else:

            self.cur\_ind -= 1

    # <variables> ::= <def\_const> <def\_variables> | <def\_const> | <def\_variables> | <def\_variables> <def\_const>

    def \_\_variables(self):

        self.\_\_def\_const()

        self.\_\_def\_variables()

        self.\_\_def\_const()

    # <start\_prog> ::= <program\_title> | <variables> | <program\_title> <variables>

    def \_\_start\_prog(self):

        self.\_\_program\_title()

        self.\_\_variables()

    def \_\_assign\_address(self):

        tmp = self.cur\_ind

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'expression\'', 400)

        if self.program1[self.cur\_ind][0] == '@':

            if self.\_\_id(adr=True):

                self.cur\_ind += 1

                if self.cur\_ind < len(self.program1):

                    if self.program1[self.cur\_ind] == ';':

                        self.cur\_ind -= 1

                        return True

        self.cur\_ind = tmp

        return False

    # <assign> ::= <id> := <expr> | <id> := @<id> | <id> := nil | <id> := <id>^

    def \_\_assign(self, if\_stmt=False):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \':=\'', 399)

        if self.program1[self.cur\_ind] != ':=':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \':=\'', 401)

        if self.\_\_assign\_address():

            return

        self.\_\_math\_expr()

    def \_\_empty\_id\_list(self):

        tmp = self.cur\_ind

        tmp += 1

        if tmp < len(self.program1):

            if self.program1[tmp] == ')':

                self.cur\_ind = tmp - 1

                return True

        return False

    # <read> ::= read(<id\_list>) | read()

    def \_\_read(self, if\_stmt=False):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'(\'', 403)

        if self.program1[self.cur\_ind] != '(':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'(\'', 405)

        if not self.\_\_empty\_id\_list():

            self.\_\_id\_list()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \')\'', 411)

        if self.program1[self.cur\_ind] != ')':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \')\'', 405)

    # <write> ::= write(<id\_list>) | write()

    def \_\_write(self, if\_stmt=False):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'(\'', 420)

        if self.program1[self.cur\_ind] != '(':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'(\'', 42)

        if not self.\_\_empty\_id\_list():

            self.\_\_id\_list(True, True, True)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \')\'', 428)

        if self.program1[self.cur\_ind] != ')':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \')\'', 430)

    # <mult\_op> ::= \* | div | and

    def \_\_mult\_op(self):

        return self.program1[self.cur\_ind] in ['\*', 'div', 'and']

    # <add\_op> ::= + | - | or | xor

    def \_\_add\_op(self):

        return self.program1[self.cur\_ind] in ['+', '-', 'or', 'xor']

    # <rel\_op> ::= = | <> | < | <= | >= | >

    def \_\_rel\_op(self):

        return self.program1[self.cur\_ind] in ['=', '<>', '<', '<=', '>=', '>']

    # <simple\_logic\_expr> ::= <term2>[<adding\_operator> <term2>]\*

    def \_\_simple\_logic\_expr(self):

        self.\_\_term2()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 530)

        if self.\_\_add\_op():

            self.\_\_term2()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 538)

            if self.\_\_add\_op():

                self.\_\_simple\_logic\_expr()

            else:

                self.cur\_ind -= 1

        else:

            self.cur\_ind -= 1

    # <term2> ::= <factor2>[<mult\_operator> <factor2>]\*

    def \_\_term2(self):

        self.\_\_factor2()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 534)

        if self.\_\_mult\_op():

            self.\_\_factor2()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 542)

            if self.\_\_mult\_op():

                self.\_\_term2()

            else:

                self.cur\_ind -= 1

        else:

            self.cur\_ind -= 1

    # <factor2> ::= <id> | <integer> | <boolean> | ( <logic\_expr> ) | not <factor2>

    def \_\_factor2(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 576)

        if self.program1[self.cur\_ind] == '(':

            self.\_\_logic\_expr()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \')\'', 582)

            if self.program1[self.cur\_ind] != ')':

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \')\'', 582)

        elif self.program1[self.cur\_ind] == 'not':

            self.\_\_factor2()

        else:

            if not (self.\_\_id(pointer=True) or self.\_\_integer()):

                self.\_\_throw\_error(self.cur\_ind, f'Incorrect expression', 589)

    # <logic\_expr> ::= <simple\_logic\_expr>[<rel\_op> <simple\_logic\_expr>]

    def \_\_logic\_expr(self):

        self.\_\_simple\_logic\_expr()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 534)

        if self.\_\_rel\_op():

            self.log\_fl = True

            self.\_\_simple\_logic\_expr()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 542)

            if self.\_\_rel\_op():

                self.\_\_logic\_expr()

            else:

                self.cur\_ind -= 1

        else:

            self.cur\_ind -= 1

    # <if> ::= if <logic\_expr> then <body> | if <logic\_expr> then <body> else <body>

    def \_\_if(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 537)

        self.cur\_ind -= 1

        self.\_\_logic\_expr()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'then\'', 543)

        if self.program1[self.cur\_ind] != 'then':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'then\'', 543)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected body of operator \'if\'', 549)

        if self.program1[self.cur\_ind] == ';':

            self.cur\_ind -= 1

            return

        elif self.program1[self.cur\_ind] == 'else':

            self.cur\_ind -= 1

            return

        else:

            self.flag = True

            self.cur\_ind -= 1

            self.\_\_body()

    def \_\_else(self):

        if self.program1[self.cur\_ind] != 'else':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected operator', 654)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 537)

        if self.program1[self.cur\_ind] == ';':

            return

        else:

            self.cur\_ind -= 1

            self.\_\_body()

    def \_\_body(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 672)

        if self.program1[self.cur\_ind] == 'begin':

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 573)

            self.cur\_ind = self.cur\_ind - 1

            self.\_\_code()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 467)

            if self.program1[self.cur\_ind] != 'end':

                self.cur\_ind -= 1

                self.\_\_code()

            else:

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 481)

        else:

            self.\_\_stmt(if\_stmt=True)

    # <index\_expr> ::= <id> := <math\_expr> to <math\_expr> | <id> := <math\_expr> downto <math\_expr>

    def \_\_index\_expr(self):

        if self.\_\_id(pointer=True):

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \':=\'', 439)

            if self.program1[self.cur\_ind] != ':=':

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \':=\'', 441)

            self.\_\_math\_expr()

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'to\downto\'', 447)

            if self.program1[self.cur\_ind] not in ['to', 'downto']:

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'to\downto\'', 449)

            self.\_\_math\_expr()

        else:

            self.\_\_throw\_error(self.cur\_ind, f'Incorrect identificator', 437)

    # <for> ::= for <index\_expr> do <body>

    def \_\_for(self):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected expression', 442)

        self.\_\_index\_expr()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'do\'', 448)

        if self.program1[self.cur\_ind] != 'do':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'do\'', 448)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected body in cycle for', 536)

        if self.program1[self.cur\_ind] == ';':

            self.cur\_ind = self.cur\_ind

        else:

            self.cur\_ind -= 1

            self.\_\_code()

    # <stmt> ::= <assign>; | <read>; | <write>; | <if> | <for>

    def \_\_stmt(self, if\_stmt=False):

        fl = False

        flag\_of\_else = False

        flag\_of\_if = False

        if self.\_\_id():                              # assign

            self.\_\_assign()

        elif self.program1[self.cur\_ind] == 'read':  # read

            self.\_\_read()

        elif self.program1[self.cur\_ind] == 'write': # write

            self.\_\_write()

        elif self.program1[self.cur\_ind] == 'if':    # if

            flag\_of\_if = True

            self.\_\_if()

        elif self.program1[self.cur\_ind] == 'for':   # for

            fl = True

            self.\_\_for()

        else:

            self.\_\_throw\_error(self.cur\_ind, f'Undefined operator', 454)

        if not flag\_of\_if:

            self.cur\_ind =  self.cur\_ind + 1 if not fl else self.cur\_ind

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 510)

            if self.program1[self.cur\_ind] != ';' and if\_stmt:

                self.\_\_else()

                flag\_of\_else = True

            if self.program1[self.cur\_ind] != ';' and not flag\_of\_else:

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \';\'', 512)

    def \_\_check\_begin\_end(self):

        tmp = self.cur\_ind

        if self.program1[tmp] == 'begin':

            tmp += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(tmp - 1, f'Found end of file but expected operator', 573)

            if self.program1[tmp] == 'end':

                tmp += 1

                if tmp >= len(self.program1):

                    self.\_\_throw\_error(tmp - 1, f'Found end of file but expected \';\'', 568)

                if self.program1[tmp] != ';':

                    self.\_\_throw\_error(tmp, f'Found \'{self.program1[tmp]}\' but expected \';\'', 571)

                else:

                    self.cur\_ind = tmp

                    return True

        return False

    # <code> ::= <stmt> | <code> <stmt> | begin end; | begin <code> end;

    # Избавимся от левой рекурсии:

    # <code> ::= <stmt> | <stmt> <code> | begin end; | | begin end; <code> | begin <code> end;

    def \_\_code(self, if\_stmt=False):

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 565)

        self.cur\_ind -= 1

        while True:

            self.cur\_ind += 1

            if not self.\_\_check\_begin\_end():

                break

        if self.program1[self.cur\_ind] == 'begin':

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 573)

            self.cur\_ind = self.cur\_ind - 1

            self.\_\_code(if\_stmt)

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 467)

            if self.program1[self.cur\_ind] != 'end':

                self.cur\_ind -= 1

                self.\_\_code(if\_stmt)

            else:

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \';\'', 473)

                if self.program1[self.cur\_ind] != ';':

                    self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \';\'', 475)

        else:

            if self.program1[self.cur\_ind] != 'end':

                self.\_\_stmt(if\_stmt)

            else:

                self.cur\_ind -= 1

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 481)

            if self.program1[self.cur\_ind] != 'end':

                self.cur\_ind -= 1

                self.\_\_code(if\_stmt)

            else:

                self.cur\_ind -= 1

    # <main> ::= begin end. | begin <code> end.

    def \_\_main(self):

        if self.program1[0] == 'begin':

            self.cur\_ind = 0

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 492)

            # Внутри begin end может быть сколько угодно точек с запятой.

            if self.program1[self.cur\_ind] == ';':

                while self.cur\_ind < len(self.program1) and self.program1[self.cur\_ind] == ';':

                    self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 501)

            tmp = self.cur\_ind

            if self.program1[self.cur\_ind] == 'end':

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'.\'', 498)

                if self.program1[self.cur\_ind] != '.':

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'.\'', 500)

                if self.program1[self.cur\_ind] == '.':

                    return

            self.cur\_ind = tmp - 1

            self.\_\_code()

        else:

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected begin', 511)

            if self.program1[self.cur\_ind] != 'begin':

                self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected begin', 514)

            self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 518)

            # Внутри begin end может быть сколько угодно точек с запятой.

            if self.program1[self.cur\_ind] == ';':

                while self.cur\_ind < len(self.program1) and self.program1[self.cur\_ind] == ';':

                    self.cur\_ind += 1

            if self.cur\_ind >= len(self.program1):

                self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected operator', 535)

            tmp = self.cur\_ind

            if self.program1[self.cur\_ind] == 'end':

                self.cur\_ind += 1

                if self.cur\_ind >= len(self.program1):

                    self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'.\'', 524)

                if self.program1[self.cur\_ind] != '.':

                    self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'.\'', 526)

                if self.program1[self.cur\_ind] == '.':

                    return

            self.cur\_ind = tmp - 1

            self.\_\_code()

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'end\'', 535)

        if self.program1[self.cur\_ind] != 'end':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'end\'', 537)

        self.cur\_ind += 1

        if self.cur\_ind >= len(self.program1):

            self.\_\_throw\_error(self.cur\_ind - 1, f'Found end of file but expected \'.\'', 540)

        if self.program1[self.cur\_ind] != '.':

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but expected \'.\'', 542)

    # <program> ::= <start\_prog> <main> | <main>

    def \_\_program(self):

        if self.program1[self.cur\_ind] not in ['program', 'var', 'const', 'begin']:

            self.\_\_throw\_error(self.cur\_ind, f'Found \'{self.program1[self.cur\_ind]}\' but declaration section or begin expected', 547)

        self.\_\_start\_prog()

        self.\_\_main()

        if self.cur\_ind + 1 < len(self.program1):

            self.\_\_throw\_error(self.cur\_ind + 1, f'Found \'{self.program1[self.cur\_ind + 1]}\' but expected end of file', 551)

    def check(self):

        if not self.program1: # пустой список

            self.error\_line = 1

            self.\_\_throw\_error(self.error\_line, f'Found end of file but declaration section or begin expected', 578)

        self.cur\_ind = 0

        self.error\_ind = 0

        self.error\_line = 0

        self.\_\_program()

Файл my\_parser.py (класс парсера):

class Parser:

    # program\_text - текст исходной программы

    def \_\_init\_\_(self, program\_text):

        self.program\_text = program\_text

    # Преобразование в массив строк без переносов строки

    def to\_string\_list1(self):

        words = []

        word = ''

        text = self.program\_text.lower()

        text = text.replace(':=', '$')

        text = text.replace('<>', '?')

        text = text.replace('>=', '!')

        text = text.replace('<=', '%')

        counter = 0

        sign = ''

        for i, letter in enumerate(text):

            if letter.isdigit() or letter.isalpha() or letter == '@' or letter == '^' or letter in ['+', '-']:

                if letter in ['+', '-']:

                    counter += 1

                    if counter == 1:

                        sign += letter

                    else:

                        word += letter

                else:

                    word += letter

            else:

                if counter == 1:

                    words.append(sign + word)

                    sign = ''

                else:

                    words.append(sign)

                    sign = ''

                if counter != 1:

                    words.append(word)

                if letter:

                    words.append(letter)

                word = ''

        words = list(filter(lambda item: item != '\t' and item != '\n' and item != '' and item != ' ', [':=' if x == '$' else x for x in words]))

        for i, item in enumerate(words):

            if item == '?': words[i] = '<>'

            if item == '!': words[i] = '>='

            if item == '%': words[i] = '<='

        return words

    # Преобразование в массив строк с переносом строк

    def to\_string\_list2(self):

        words = []

        word = ''

        text = self.program\_text.lower()

        text = text.replace(':=', '$')

        text = text.replace('<>', '?')

        text = text.replace('>=', '!')

        text = text.replace('<=', '%')

        counter = 0

        sign = ''

        for i, letter in enumerate(text):

            if letter.isdigit() or letter.isalpha() or letter == '@' or letter == '^' or letter in ['+', '-']:

                if letter in ['+', '-']:

                    counter += 1

                    if counter == 1:

                        sign += letter

                    else:

                        word += letter

                else:

                    word += letter

            else:

                if counter == 1:

                    words.append(sign + word)

                    sign = ''

                else:

                    words.append(sign)

                    sign = ''

                if counter != 1:

                    words.append(word)

                if letter:

                    words.append(letter)

                word = ''

        words = list(filter(lambda item: item != '\t' and item != '' and item != ' ', [':=' if x == '$' else x for x in words]))

        for i, item in enumerate(words):

            if item == '?': words[i] = '<>'

            if item == '!': words[i] = '>='

            if item == '%': words[i] = '<='

        return words