МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КУБГУ»)

Кафедра вычислительных технологий

Отчёт

по лабораторной работе №1

по дисциплине «Методы разработки трансляторов»

на тему: «Построение лексического анализатора»

Работу выполнил

студент группы

ФИ 31/2

Костров А.А.

_____

(подпись)

Преподаватель:

Усов П.Е.

_____

(подпись)

Краснодар
2026

Цель работы. Требуется построить лексический анализатор заданного подмножества языковых конструкций и операторов входного языка программирования – языка R.

Задание. Необходимо построить программу лексического анализатора, корректно распознающего все заданные лексемы и формирующего таблицы лексем и внутреннее представление проанализированного текста. На вход программы подаётся файл, содержащий текст программы на входном языке программирования – языке R. Результатом работы программы должен быть файл, содержащий последовательность кодов лексем входной программы, а также один или несколько файлов, содержащие все таблицы лексем.

Ход работы:

1 Описание подмножества языковых конструкций и операторов входного языка

Идентификаторы- в языке R представляют собой произвольные последовательности букв, цифр, точек и символов подчеркивания, начинающиеся с буквы или точки (если за точкой следует не цифра). Примеры: calculate_stats, data, mean_val

Числовые константы целого типа- представляют собой произвольные последовательности цифр без знака. Примеры: 42, 0, 1000000

Числовые константы вещественного типа с фиксированной точкой- представляют собой последовательности цифр, включающие одну десятичную точку. Примеры: 123.45, 0.5, 3.14159, 12.34

Числовые константы вещественного типа с плавающей точкой- представляют собой последовательности, включающие цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» (необязательный). Примеры: 2.5e-3, 1.6E-19, 3e5, 123.e-4

Символьные (строковые) константы- представляют собой последовательности символов, заключенные в двойные или одинарные кавычки. Поддерживаются escape-последовательности. Примеры: "Hello World", 'test string', "escaped \"quotes\""

Комментарии- – однострочные, начинаются с символа «#» и продолжаются до конца строки. Пример: # Это комментарий в R

Арифметические операции- : сложение («+»); вычитание («-»); умножение («*»); деление («/»); возведение в степень («^» или «**»); остаток от деления («%%»); целочисленное деление («%/%»); матричное умножение («%*%»).

Операции сравнения- : меньше («<»); больше («>»); равно («==»); не равно («!=»); меньше или равно («<=»); больше или равно («>=»); проверка вхождения («%in%»).

Оператор присваивания- имеет вид «<-» (основной), «->» (обратный), «=» (альтернативный), «<<-» и «->>» (присваивание в родительское окружение). Примеры: x <- 10, y -> z, a = 5

Логические операции- : И («&» и «&&»); ИЛИ («|» и «||»); НЕ («!»); исключающее ИЛИ («xor()»).

Операторы управления потоком- : условный оператор «if ... else», циклы «for», «while», «repeat», операторы прерывания «break» и «next».

Оператор определения функции- начинается с ключевого слова «function», за которым в круглых скобках следуют параметры, затем тело функции в фигурных скобках. Пример: function(x, y) { return(x + y) }

Обращение к функциям- – идентификатор функции, после которого в круглых скобках следует список аргументов, разделенных запятыми. Аргументы могут быть именованными (name = value). Примеры: mean(data), c(1, 2, 3)

Операторы доступа к элементам- : «$» для доступа по имени в списке/датафрейме; «[» и «]» для индексации; «[[» и «]]» для извлечения одного элемента; «@» для доступа к слотам S4-объектов.

2 Таблицы, используемые лексическим анализатором

Таблица служебных слов:

№	Слово
1	FALSE
2	Inf
3	NA
4	NULL
5	NaN
6	TRUE
7	aggregate
8	anova
9	apply
10	break
11	c
12	cat
13	cbind
14	else
15	for
16	function
17	getwd
18	ggplot
19	glm
20	if
21	in
22	lapply
23	length
24	library
25	list
26	lm
27	mapply
28	matrix
29	merge

№	Слово
30	ncol
31	next
32	nrow
33	paste
34	plot
35	predict
36	print
37	rbind
38	repeat
39	require
40	return
41	sapply
42	setwd
43	source
44	sprintf
45	stop
46	subset
47	summary
48	switch
49	tapply
50	try
51	tryCatch
52	warning
53	while

Таблица разделителей:

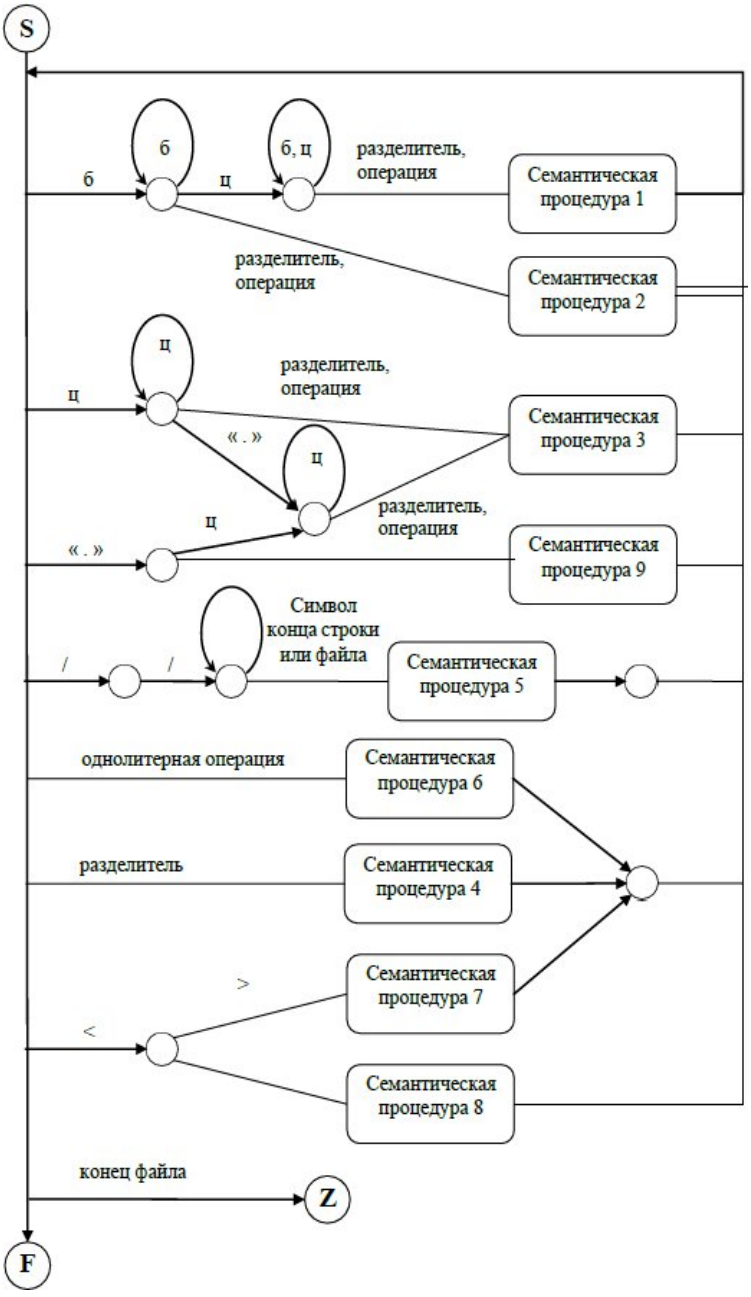
№	Разделитель
1	,
2	;
3	:
4	::
5	:::
6	(
7	)
8	[
9	]
10	[[
11	]]
12	{
13	}
14	`
15	$
16	@
17	\

Таблица операций:

№	Операция
1	+
2	-
3	*
4	/
5	^
6	**
7	%%
8	%/%
9	%*%
10	%in%
11	<
12	>
13	<=
14	>=
15	==
16	!=
17	=
18	<-
19	<<-
20	->
21	->>
22	&
23	|
24	!
25	&&
26	||
27	~
28	:
29	$
30	@

№	Операция
31	%>%
32	%T>%
33	%<>%
34	%$%

3 Диаграмма состояний сканера



На данной диаграмме состояния сканера представлены вершинами, а переходы между состояниями - дугами (направленными линиями). Каждый переход связан с чтением очередного символа из текста входной программы. Поэтому дуга взвешена (помечена) символом или множеством символов, которые вызывают данный переход. Если в диаграмме состояний есть невзвешенная дуга, ведущая из какого-либо состояния, то считается, что она взвешена любыми символами кроме тех, которыми взвешены другие дуги, исходящие из данного

состояния. Это позволяет не перегружать диаграмму лишними символами. Скруглённый прямоугольник в разрыве дуги указывает на семантическую процедуру, выполняемую при данном переходе. Если переход не сопровождается семантической процедурой, то текущий символ добавляется к буферу, в котором формируется лексема. Работа каждой семантической процедуры завершается очищением буфера.

Сканер всегда содержит три стандартных состояния:

- S – начальное состояние сканера. Лексический разбор всегда начинается из этого состояния;
- Z – заключительное состояние сканера. Если в процессе разбора достигнуто данное состояние, это означает, что разбор успешно завершен;
- F – состояние ошибки. Если встретился символ, не входящий во входной алфавит, то сканер переходит в состояние ошибки и прекращает разбор. Кроме того, если в каком-либо состоянии на входе появился символ, по которому не предусмотрен переход из этого состояния, сканер также переходит в состояние ошибки (на диаграмме эти переходы не изображаются, чтобы избежать лишнего загромождения рисунка).

За семантическими процедурами процессора закреплены следующие функции.

Семантическая процедура 1: провести поиск сформированного слова в таблице идентификаторов. Если такое слово в таблице идентификаторов не найдено, то занести сформированное слово в таблицу идентификаторов. Сформировать и выдать в выходную последовательность лексему идентификатора во внутреннем представлении.

Семантическая процедура 2: провести поиск сформированного слова в таблице служебных слов. Если такое слово в таблице служебных слов не найдено, то выполнить Семантическую процедуру 1, иначе сформировать и выдать в выходную последовательность лексему служебного слова во внутреннем представлении.

Семантическая процедура 3: занести сформированное слово в таблицу констант. Сформировать и выдать в выходную последовательность лексему константы во внутреннем представлении.

Семантическая процедура 4: провести поиск текущего символа в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении.

Семантическая процедура 5: удалить сформированную последовательность символов.

Семантическая процедура 6: провести поиск текущего символа в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 7: добавить текущий символ к сформированному слову. Провести поиск сформированного слова в

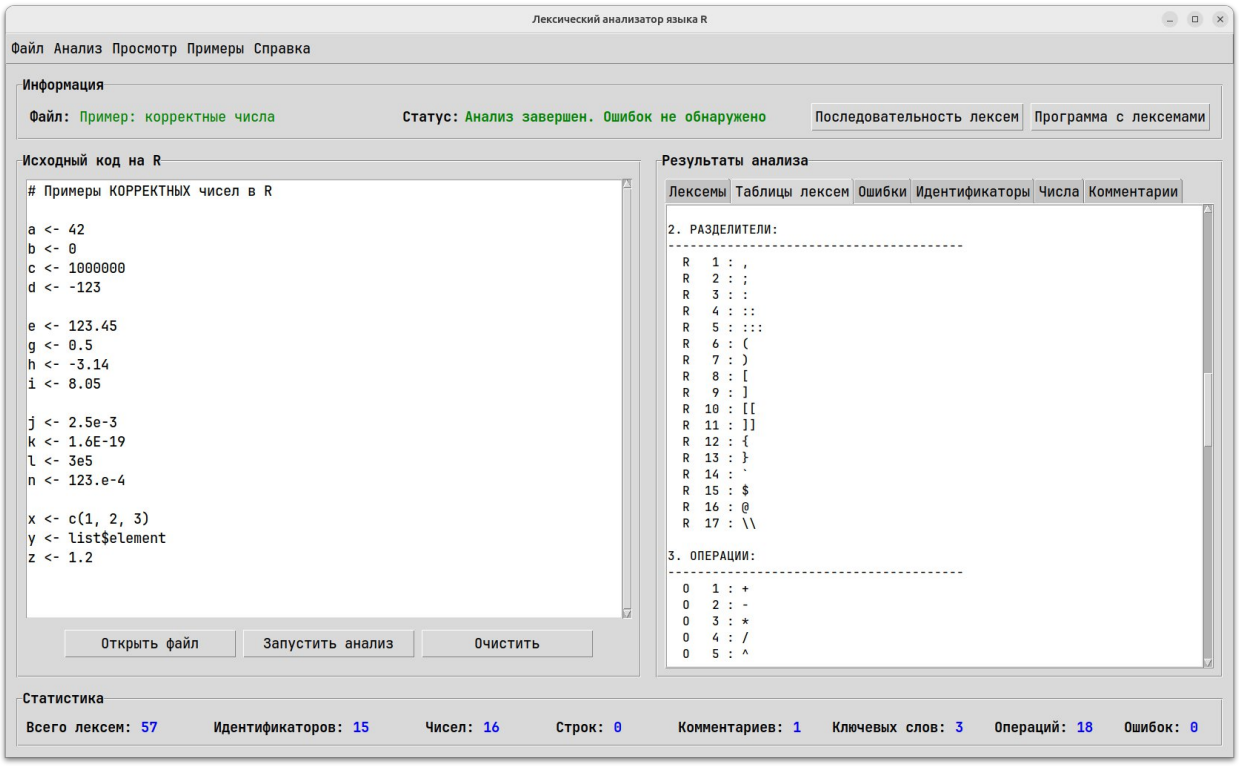
таблице операций. Если такое слово в таблице операций не найдено, то перейти в состояние ошибки. Если поиск успешен, то сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 8: провести поиск сформированного слова в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

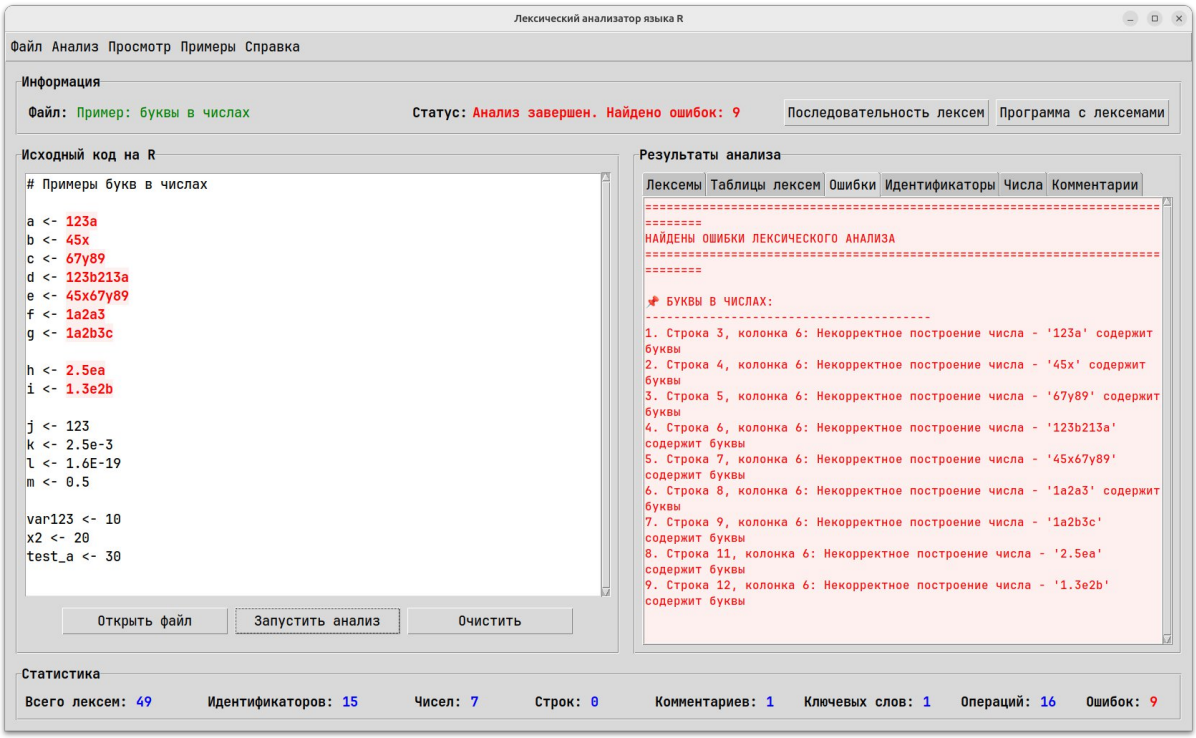
Семантическая процедура 9: провести поиск сформированного слова в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении.

4 Программа

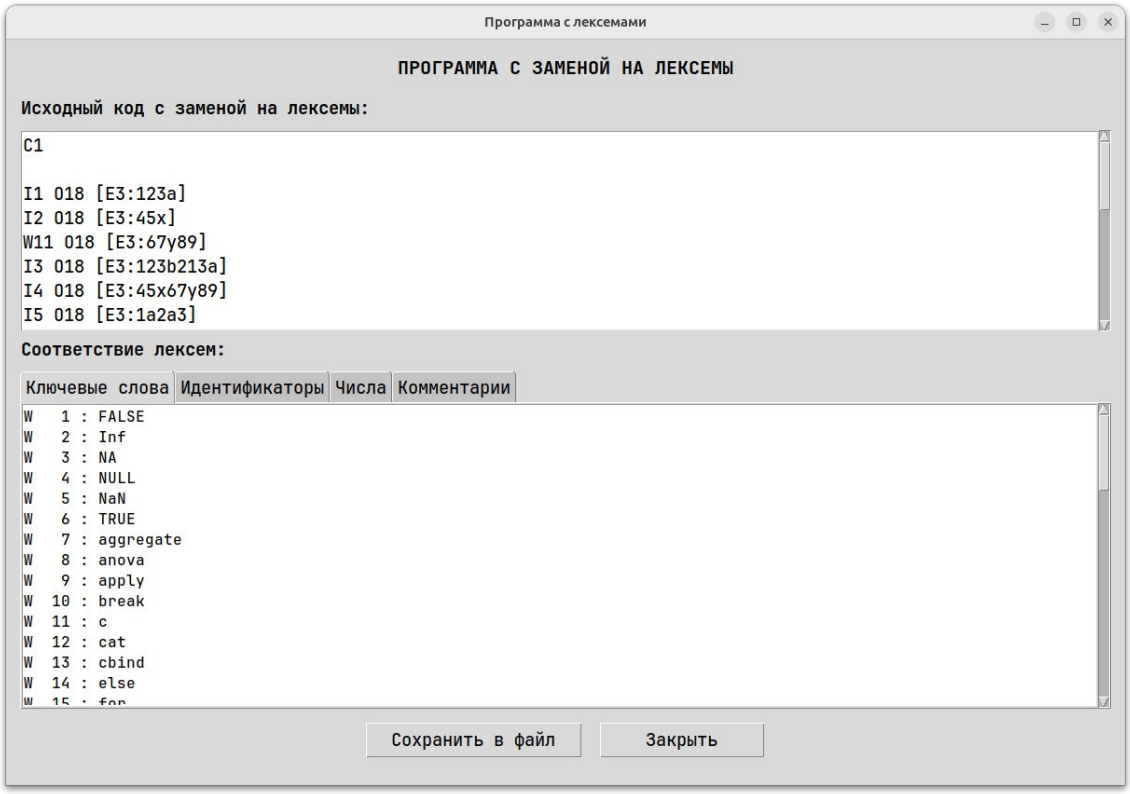
Программа представляет собой приложение с графическим интерфейсом. В левой части окна находится панель ввода исходного кода на языке R, а справа находятся результаты анализа.



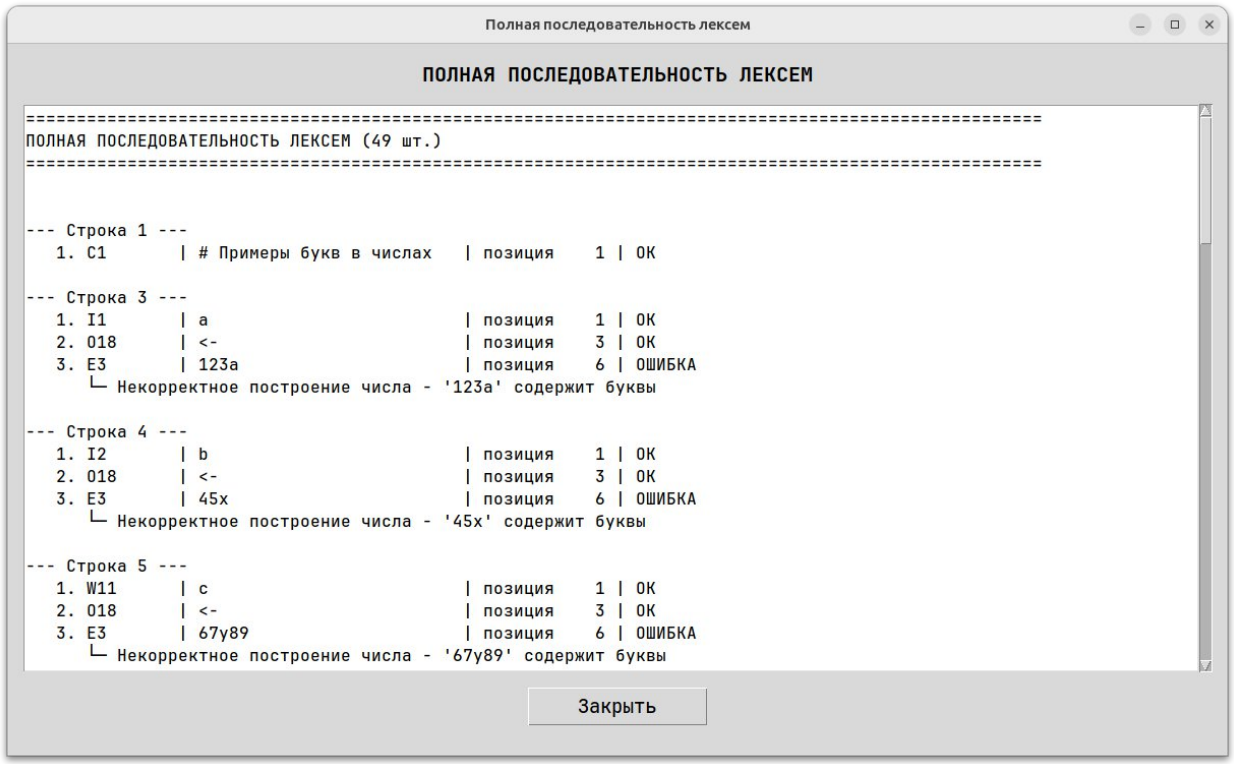
Если в ходе лексического анализа будут найдены ошибки, то подробная информация о них отобразится в правой панели.



При нажатии на кнопку «Программа с лексемами» появится окно с кодом на языке лексем.



При нажатии на кнопку «Последовательность лексем» отображается подробная информация о всех распознанных лексемах.



Программа состоит из следующих основных компонентов:

Компонент	Назначение
RLexer	Ядро лексического анализатора
RLexerGUI	Графический интерфейс пользователя
Token	Класс для представления лексемы
LexemType	Перечисление типов лексем
HelpWindow	Вспомогательное окно справки

Класс RLexer

Класс RLexer является ядром анализатора и содержит основную логику лексического анализа. Он отвечает за разбор входного текста и формирование последовательности токенов.

Таблицы лексем

Класс содержит следующие таблицы для хранения лексем:

```
keywords     - служебные слова языка R (55 элементов)
delimiters   - разделители (24 элемента)
operations   - операции (34 элемента)
identifiers  - идентификаторы (динамическая таблица)
numbers      - числовые литералы (динамическая таблица)
strings      - строковые литералы (динамическая таблица)
comments     - комментарии (динамическая таблица)
```

Метод tokenize

Основной метод класса, выполняющий лексический анализ. Алгоритм работы:

1. Пропуск пробельных символов
2. Обработка комментариев (начинающихся с #)
3. Обработка строковых литералов
4. Обработка числовых литералов
5. Обработка идентификаторов и ключевых слов
6. Обработка операций и разделителей

Класс RLexerGUI

Класс RLexerGUI реализует графический интерфейс пользователя на базе библиотеки tkinter. Интерфейс предоставляет следующие возможности:

- Редактор исходного кода с подсветкой синтаксиса
- Отображение результатов анализа в нескольких вкладках
- Загрузка кода из файла
- Сохранение результатов анализа
- Встроенные примеры кода

Вкладки результатов

Результаты анализа отображаются в следующих вкладках:

Вкладка	Содержимое
Лексемы	Последовательность распознанных лексем
Таблицы лексем	Таблицы всех типов лексем
Ошибки	Список обнаруженных ошибок
Идентификаторы	Таблица идентификаторов
Числа	Таблица числовых литералов с типами
Комментарии	Таблица комментариев

Класс Token

Класс Token (датакласс) представляет отдельную лексему и содержит следующие поля:

```
code     - код лексемы (например, I1, W5, N3)
value    - значение лексемы (исходный текст)
line     - номер строки в исходном коде
column   - номер колонки в исходном коде
lex_type - тип лексемы (из перечисления LexemType)
error_msg - сообщение об ошибке (если есть)
```

Перечисление LexemType

Перечисление определяет типы лексем:

```
KEYWORD    = "W"   - служебные слова
DELIMITER  = "R"   - разделители
OPERATION  = "O"   - операции
IDENTIFIER = "I"   - идентификаторы
NUMBER     = "N"   - числа
STRING     = "S"   - строки
COMMENT    = "C"   - комментарии
ERROR      = "E"   - ошибки
```

Полный текст разработанной программы:

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, scrolledtext
from enum import Enum
from dataclasses import dataclass
from pathlib import Path
import re
import datetime


def main():
    root = tk.Tk()
    root.option_add('*Font', 'TkDefaultFont 14')
    app = RLexerGUI(root)
    app.load_example("correct")
    root.mainloop()


class LexemType(Enum):
    KEYWORD = "W"
    DELIMITER = "R"
    OPERATION = "O"
    IDENTIFIER = "I"
    NUMBER = "N"
    STRING = "S"
    COMMENT = "C"
    ERROR = "E"


@dataclass
class Token:
    code: str
    value: str
    line: int
    column: int
    lex_type: LexemType
    error_msg: str = ""


class RLexer:
    def __init__(self):
        self.keywords = {}
        self.delimiters = {}
        self.operations = {}
        self.identifiers = {}
        self.numbers = {}
        self.strings = {}
        self.comments = {}
        self.token_sequence = []
        self.errors = []
        self.error_tokens = []
        self.current_line = 1
        self.current_column = 1
        self.original_code = ""
        self._init_tables()

    def _init_tables(self):
        keywords_list = [
            "if", "else", "while", "for", "in", "next", "break",
            "function", "return", "TRUE", "FALSE", "NULL", "NA",
            "Inf", "NaN", "repeat", "switch", "try", "tryCatch",
            "stop", "warning", "require", "library", "source",
            "setwd", "getwd", "list", "matrix",
            "c", "cbind", "rbind", "length", "nrow", "ncol",
            "summary", "print", "cat", "paste", "sprintf",
            "subset", "merge", "apply", "lapply", "sapply",
            "tapply", "mapply", "aggregate", "plot", "ggplot",
            "lm", "glm", "anova", "predict"
        ]
        for i, kw in enumerate(sorted(keywords_list), 1):
            self.keywords[kw] = i

        delimiters_list = [
            ",", ";", ":", "::", ":::",
            "(", ")", "[", "]", "[[", "]]",
            "{", "}", "`", "$", "@", "\\"
        ]
        for i, delim in enumerate(delimiters_list, 1):
            self.delimiters[delim] = i

        operations_list = [
            "+", "-", "*", "/", "^", "**",
            "%%", "%/%", "%*%", "%in%",
            "<", ">", "⩽", "⩾", "⩵", "≠",
            "=", "←", "↞", "->", "->>",
            "&", "|", "!", "&&", "||",
            "~", ":", "$", "@",
```

```
        "%>%", "%T>%", "%<>%", "%$%"
    ]
    for i, op in enumerate(operations_list, 1):
        self.operations[op] = i

def reset(self):
    self.identifiers = {}
    self.numbers = {}
    self.strings = {}
    self.comments = {}
    self.token_sequence = []
    self.errors = []
    self.error_tokens = []
    self.current_line = 1
    self.current_column = 1

def is_keyword(self, word):
    return self.keywords.get(word)

def is_delimiter(self, char):
    return self.delimiters.get(char)

def is_operation(self, op):
    return self.operations.get(op)

def add_identifier(self, name):
    if name not in self.identifiers:
        self.identifiers[name] = len(self.identifiers) + 1
    return self.identifiers[name]

def add_number(self, num):
    if num not in self.numbers:
        self.numbers[num] = len(self.numbers) + 1
    return self.numbers[num]

def add_string(self, string):
    if string not in self.strings:
        self.strings[string] = len(self.strings) + 1
    return self.strings[string]

def add_comment(self, comment):
    if comment not in self.comments:
        self.comments[comment] = len(self.comments) + 1
    return self.comments[comment]

def is_valid_number(self, num_str):
    if not num_str:
        return False, "Пустое число"

    if '..' in num_str:
        return False, "Множественное использование точки"

    if num_str.count('.') > 1:
        if 'e' in num_str.lower() or 'E' in num_str.lower():
            parts = re.split(r'[eE]', num_str)
            if len(parts) == 2:
                mantissa = parts[0]
                exponent = parts[1]
                if mantissa.count('.') > 1:
                    return False, "Некорректное использование точки - несколько десятичных разделителей в мантиссе"
                if '.' in exponent:
                    return False, "Некорректное использование точки - точка в экспоненте"
            else:
                return False, "Некорректная экспоненциальная запись"
        else:
            return False, "Некорректное использование точки - число содержит несколько десятичных разделителей"

    for i, char in enumerate(num_str):
        if char.isalpha():
            if char.lower() == 'e' and i > 0 and i < len(num_str) - 1:
                next_char = num_str[i + 1]
                if next_char.isdigit() or next_char in '+-':
                    continue
            return False, f"Некорректное построение числа - содержит букву '{char}'"

    if 'e' in num_str.lower() or 'E' in num_str.lower():
        parts = re.split(r'[eE]', num_str)
        if len(parts) != 2:
            return False, "Некорректная экспоненциальная запись"
        if not parts[0] or not parts[1]:
            return False, "Некорректная экспоненциальная запись"

        mantissa = parts[0]
        if mantissa.count('.') > 1:
            return False, "Некорректное использование точки - несколько десятичных разделителей в мантиссе"

        exponent = parts[1]
        if '.' in exponent:
            return False, "Некорректное использование точки - точка в экспоненте"
```

```
        if exponent and exponent[0] in '+-':
            exponent = exponent[1:]
        if not exponent or not exponent.isdigit():
            return False, "Некорректная экспоненциальная запись"

    return True, "Корректное число"

def tokenize(self, code):
    self.reset()
    self.original_code = code

    i = 0
    length = len(code)

    while i < length:
        char = code[i]
        start_column = self.current_column

        if char.isspace():
            if char == '\n':
                self.current_line += 1
                self.current_column = 1
            else:
                self.current_column += 1
            i += 1
            continue

        if char == '#':
            comment = ''
            start_line = self.current_line
            while i < length and code[i] != '\n':
                comment += code[i]
                i += 1
                self.current_column += 1
            token_id = self.add_comment(comment)
            self.token_sequence.append(Token(f"C{token_id}", comment, start_line, start_column, LexemType.COMMENT))
            continue

        if char in ('"', "'"):
            quote = char
            string = char
            i += 1
            self.current_column += 1
            start_line = self.current_line

            while i < length and code[i] != quote:
                if code[i] == '\\' and i + 1 < length:
                    string += code[i] + code[i + 1]
                    i += 2
                    self.current_column += 2
                else:
                    string += code[i]
                    i += 1
                    self.current_column += 1

            if i < length and code[i] == quote:
                string += quote
                i += 1
                self.current_column += 1
                token_id = self.add_string(string)
                self.token_sequence.append(Token(f"S{token_id}", string, start_line, start_column, LexemType.STRING))
            else:
                error_msg = f"Незакрытая строка"
                self.errors.append(f"Строка {start_line}, колонка {start_column}: {error_msg}")
                self.token_sequence.append(Token("E1", string, start_line, start_column, LexemType.ERROR, error_msg))
            continue

        if char == '.' and i + 2 < length and code[i + 1] == '.' and code[i + 2] == '.':
            start_line = self.current_line
            start_col = self.current_column

            prev_is_digit = i > 0 and code[i - 1].isdigit()
            j = i + 3
            dot_seq = "..."
            while j < length and (code[j].isalnum() or code[j] == '.'):
                dot_seq += code[j]
                j += 1

            if prev_is_digit:
                error_msg = f"Множественное использование точки"
                self.errors.append(f"Строка {start_line}, колонка {start_col}: {error_msg} - '{dot_seq}'")
                self.token_sequence.append(Token("E4", dot_seq, start_line, start_col, LexemType.ERROR, error_msg))
                i = j
                self.current_column = start_col + len(dot_seq)
            else:
                token_id = self.add_identifier(dot_seq)
                self.token_sequence.append(Token(f"I{token_id}", dot_seq, start_line, start_col, LexemType.IDENTIFIER))
                i = j
                self.current_column = start_col + len(dot_seq)
            continue
```

```
if char == '.' and i + 1 < length and code[i + 1] == '.':
    start_line = self.current_line
    start_col = self.current_column

    prev_is_digit = i > 0 and code[i - 1].isdigit()
    j = i + 2
    dot_seq = ".."
    while j < length and (code[j].isalnum() or code[j] == '.'):
        dot_seq += code[j]
        j += 1

    if prev_is_digit:
        error_msg = f"Множественное использование точки"
        self.errors.append(f"Строка {start_line}, колонка {start_col}: {error_msg} - '{dot_seq}'")
        self.token_sequence.append(Token("E4", dot_seq, start_line, start_col, LexemType.ERROR, error_msg))
        i = j
        self.current_column = start_col + len(dot_seq)
    else:
        token_id = self.add_identifier(dot_seq)
        self.token_sequence.append(Token(f"I{token_id}", dot_seq, start_line, start_col, LexemType.IDENTIFIER))
        i = j
        self.current_column = start_col + len(dot_seq)
    continue

if char.isdigit() or (char == '.' and i + 1 < length and code[i + 1].isdigit()):
    number = ''
    start_line = self.current_line
    start_col = self.current_column

    if char == '.':
        number += char
        i += 1
        self.current_column += 1

    while i < length:
        current_char = code[i]

        if current_char.isdigit():
            number += current_char
            i += 1
            self.current_column += 1
            continue

        elif current_char == '.':
            number += current_char
            i += 1
            self.current_column += 1
            continue

        elif current_char in '+-':
            if number and number[-1].lower() == 'e':
                number += current_char
                i += 1
                self.current_column += 1
                continue
            else:
                break

        elif current_char.isalpha():
            if current_char.lower() == 'e' and number and not number[-1].isalpha():
                if i + 1 < length and (code[i + 1].isdigit() or code[i + 1] in '+-'):
                    number += current_char
                    i += 1
                    self.current_column += 1
                    continue
            number += current_char
            i += 1
            self.current_column += 1
            while i < length and (code[i].isalnum() or code[i] == '.'):
                number += code[i]
                i += 1
                self.current_column += 1
            error_msg = f"Некорректное построение числа - '{number}' содержит буквы"
            self.errors.append(f"Строка {start_line}, колонка {start_col}: {error_msg}")
            self.token_sequence.append(Token("E3", number, start_line, start_col, LexemType.ERROR, error_msg))
            break

        else:
            break

    if number and (not self.token_sequence or self.token_sequence[-1].lex_type != LexemType.ERROR):
        is_valid, error_msg = self.is_valid_number(number)
        if is_valid:
            token_id = self.add_number(number)
            self.token_sequence.append(Token(f"N{token_id}", number, start_line, start_col, LexemType.NUMBER))
        else:
            self.errors.append(f"Строка {start_line}, колонка {start_col}: {error_msg} - '{number}'")
            self.token_sequence.append(Token("E4", number, start_line, start_col, LexemType.ERROR, error_msg))
    continue
```

```
            if char.isalpha() or char == '_' or (char == '.' and i + 1 < length and not code[i + 1].isdigit()):
                word = ''
                start_line = self.current_line
                start_col = self.current_column

                while i < length and (code[i].isalnum() or code[i] in '._'):
                    word += code[i]
                    i += 1
                    self.current_column += 1

                kw_id = self.is_keyword(word)
                if kw_id:
                    self.token_sequence.append(Token(f"W{kw_id}", word, start_line, start_col, LexemType.KEYWORD))
                else:
                    token_id = self.add_identifier(word)
                    self.token_sequence.append(Token(f"I{token_id}", word, start_line, start_col, LexemType.IDENTIFIER))
                continue

            op_found = False

            for length_op in range(4, 0, -1):
                if i + length_op <= length:
                    potential_op = code[i:i+length_op]
                    op_id = self.is_operation(potential_op)
                    if op_id:
                        self.token_sequence.append(Token(f"O{op_id}", potential_op, self.current_line, start_column,
LexemType.OPERATION))
                        i += length_op
                        self.current_column += length_op
                        op_found = True
                        break

            if not op_found:
                delim_id = self.is_delimiter(char)
                if delim_id:
                    self.token_sequence.append(Token(f"R{delim_id}", char, self.current_line, start_column, LexemType.DELIMITER))
                    i += 1
                    self.current_column += 1
                    continue

                op_id = self.is_operation(char)
                if op_id:
                    self.token_sequence.append(Token(f"O{op_id}", char, self.current_line, start_column, LexemType.OPERATION))
                    i += 1
                    self.current_column += 1
                    continue

            if not op_found and not delim_id:
                error_msg = f"Неизвестный символ '{char}'"
                self.errors.append(f"Строка {self.current_line}, колонка {self.current_column}: {error_msg}")
                self.token_sequence.append(Token("E5", char, self.current_line, start_column, LexemType.ERROR, error_msg))
                i += 1
                self.current_column += 1

        return self.token_sequence

    def generate_lexeme_program(self):
        if not self.token_sequence:
            return "Нет данных для отображения. Сначала выполните анализ."

        lines = {}
        for token in self.token_sequence:
            if token.line not in lines:
                lines[token.line] = []
            lines[token.line].append(token)

        result = []
        result.append("=" * 80)
        result.append("ПРОГРАММА С ЗАМЕНОЙ НА ЛЕКСЕМЫ")
        result.append("=" * 80)
        result.append("")

        for line_num in sorted(lines.keys()):
            line_tokens = lines[line_num]
            line_text = ""
            current_pos = 1

            for token in sorted(line_tokens, key=lambda x: x.column):
                if token.column > current_pos:
                    line_text += " " * (token.column - current_pos)

                if token.lex_type == LexemType.ERROR:
                    line_text += f"[{token.code}:{token.value}]"
                else:
                    line_text += token.code

                current_pos = token.column + len(token.value)

            result.append(f"Строка {line_num:3d}: {line_text}")
```

```
        result.append("")
        result.append("=" * 80)
        result.append("СООТВЕТСТВИЕ ЛЕКСЕМ:")
        result.append("-" * 40)

        result.append("\nКлючевые слова:")
        for word, idx in sorted(self.keywords.items(), key=lambda x: x[1])[:10]:
            result.append(f"  W{idx:4d} : {word}")

        result.append("\nИдентификаторы:")
        for name, idx in sorted(self.identifiers.items(), key=lambda x: x[1])[:10]:
            result.append(f"  I{idx:4d} : {name}")

        result.append("\nЧисла:")
        for num, idx in sorted(self.numbers.items(), key=lambda x: x[1])[:10]:
            result.append(f"  N{idx:4d} : {num}")

        result.append("\nКомментарии:")
        for comment, idx in sorted(self.comments.items(), key=lambda x: x[1])[:10]:
            result.append(f"  C{idx:4d} : {comment}")

        if len(self.keywords) > 10 or len(self.identifiers) > 10 or len(self.numbers) > 10 or len(self.comments) > 10:
            result.append("\n... и другие (см. полные таблицы)")

        return '\n'.join(result)

    def generate_clean_lexeme_program(self):
        if not self.token_sequence or not self.original_code:
            return ""

        code_lines = self.original_code.splitlines()
        token_lines = {}
        for token in self.token_sequence:
            if token.line not in token_lines:
                token_lines[token.line] = []
            token_lines[token.line].append(token)

        result = []
        for line_num in range(1, len(code_lines) + 1):
            line_tokens = token_lines.get(line_num, [])
            if line_tokens:
                line_text = ""
                current_pos = 1
                for token in sorted(line_tokens, key=lambda x: x.column):
                    if token.column > current_pos:
                        line_text += " " * (token.column - current_pos)
                    if token.lex_type == LexemType.ERROR:
                        line_text += f"[{token.code}:{token.value}]"
                    else:
                        line_text += token.code
                    current_pos = token.column + len(token.value)
                result.append(line_text)
            else:
                result.append("")

        return '\n'.join(result)


class HelpWindow:
    def __init__(self, parent, title, content, width=800, height=600):
        self.window = tk.Toplevel(parent)
        self.window.title(title)
        self.window.geometry(f"{width}x{height}")
        self.window.configure(bg='#f0f0f0')
        self.window.transient(parent)
        self.window.grab_set()

        main_frame = ttk.Frame(self.window, padding="15")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        self.window.columnconfigure(0, weight=1)
        self.window.rowconfigure(0, weight=1)
        main_frame.columnconfigure(0, weight=1)
        main_frame.rowconfigure(0, weight=1)

        text_widget = scrolledtext.ScrolledText(
            main_frame,
            wrap=tk.WORD,
            font=('TkDefaultFont', 12),
            background='#ffffff'
        )
        text_widget.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        text_widget.insert(1.0, content)
        text_widget.config(state=tk.DISABLED)

        ttk.Button(main_frame, text="Закрыть", command=self.window.destroy).grid(row=1, column=0, pady=10)


class RLexerGUI:
    def __init__(self, root):
```

```
        self.root = root
        self.root.title("Лексический анализатор языка R")
        self.root.geometry("1600x900")

        self.setup_fonts()

        self.lexer = RLexer()
        self.current_file = None

        self.setup_ui()

    def setup_fonts(self):
        from tkinter import font
        available_fonts = list(font.families())

        preferred_fonts = ['JetBrains Mono', 'Ubuntu', 'DejaVu Sans', 'Liberation Sans', 'Arial', 'Helvetica', 'TkDefaultFont']

        self.default_font = 'TkDefaultFont'
        for pref_font in preferred_fonts:
            if pref_font in available_fonts:
                self.default_font = pref_font
                break

        self.font_size = 14
        self.small_font_size = 12
        self.large_font_size = 15

        style = ttk.Style()

        style.configure('.', font=(self.default_font, self.font_size))
        style.configure('TLabel', font=(self.default_font, self.font_size))
        style.configure('TButton', font=(self.default_font, self.font_size))
        style.configure('TFrame', font=(self.default_font, self.font_size))
        style.configure('TLabelframe', font=(self.default_font, self.font_size, 'bold'))
        style.configure('TLabelframe.Label', font=(self.default_font, self.font_size, 'bold'))
        style.configure('TNotebook', font=(self.default_font, self.font_size))
        style.configure('TNotebook.Tab', font=(self.default_font, self.font_size))
        style.configure('Treeview', font=(self.default_font, self.small_font_size))
        style.configure('Treeview.Heading', font=(self.default_font, self.font_size, 'bold'))

        self.root.option_add('*Font', (self.default_font, self.font_size))
        self.root.option_add('*TLabel.Font', (self.default_font, self.font_size))
        self.root.option_add('*TButton.Font', (self.default_font, self.font_size))
        self.root.option_add('*Menu.Font', (self.default_font, self.font_size))
        self.root.option_add('*Menubutton.Font', (self.default_font, self.font_size))
        self.root.option_add('*Entry.Font', (self.default_font, self.font_size))
        self.root.option_add('*Listbox.Font', (self.default_font, self.font_size))
        self.root.option_add('*Spinbox.Font', (self.default_font, self.font_size))

    def setup_ui(self):
        MENU_FONT = (self.default_font, self.font_size)

        menubar = tk.Menu(self.root, font=MENU_FONT)
        self.root.config(menu=menubar)

        file_menu = tk.Menu(menubar, tearoff=0, font=MENU_FONT)
        menubar.add_cascade(label="Файл", menu=file_menu, font=MENU_FONT)
        file_menu.add_command(label="Открыть файл", command=self.open_file, accelerator="Ctrl+O", font=MENU_FONT)
        file_menu.add_command(label="Сохранить результат", command=self.save_results, accelerator="Ctrl+S", font=MENU_FONT)
        file_menu.add_separator()
        file_menu.add_command(label="Выход", command=self.root.quit, font=MENU_FONT)

        analyze_menu = tk.Menu(menubar, tearoff=0, font=MENU_FONT)
        menubar.add_cascade(label="Анализ", menu=analyze_menu, font=MENU_FONT)
        analyze_menu.add_command(label="Запустить анализ", command=self.analyze, accelerator="F5", font=MENU_FONT)
        analyze_menu.add_command(label="Очистить всё", command=self.clear_all, font=MENU_FONT)

        view_menu = tk.Menu(menubar, tearoff=0, font=MENU_FONT)
        menubar.add_cascade(label="Просмотр", menu=view_menu, font=MENU_FONT)
        view_menu.add_command(label="Полная последовательность лексем", command=self.show_full_sequence, font=MENU_FONT)
        view_menu.add_command(label="Программа с лексемами", command=self.show_lexeme_program, font=MENU_FONT)

        examples_menu = tk.Menu(menubar, tearoff=0, font=MENU_FONT)
        menubar.add_cascade(label="Примеры", menu=examples_menu, font=MENU_FONT)
        examples_menu.add_command(label="Корректный код R", command=lambda: self.load_example("correct"), font=MENU_FONT)
        examples_menu.add_command(label="Ошибки в числах", command=lambda: self.load_example("errors"), font=MENU_FONT)
        examples_menu.add_command(label="Множественные точки", command=lambda: self.load_example("dots"), font=MENU_FONT)
        examples_menu.add_command(label="Буквы в числах", command=lambda: self.load_example("letters"), font=MENU_FONT)
        examples_menu.add_command(label="Корректные числа", command=lambda: self.load_example("correct_numbers"), font=MENU_FONT)

        help_menu = tk.Menu(menubar, tearoff=0, font=MENU_FONT)
        menubar.add_cascade(label="Справка", menu=help_menu, font=MENU_FONT)
        help_menu.add_command(label="О программе", command=self.show_about, font=MENU_FONT)
        help_menu.add_command(label="Синтаксис R", command=self.show_r_syntax, font=MENU_FONT)
        help_menu.add_command(label="Типы ошибок", command=self.show_error_types, font=MENU_FONT)

        self.root.bind('<Control-o>', lambda e: self.open_file())
        self.root.bind('<Control-s>', lambda e: self.save_results())
        self.root.bind('<F5>', lambda e: self.analyze())
```

```
        main_frame = ttk.Frame(self.root, padding="15")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)
        main_frame.columnconfigure(0, weight=1)
        main_frame.columnconfigure(1, weight=1)
        main_frame.rowconfigure(1, weight=1)

        info_frame = ttk.LabelFrame(main_frame, text="Информация", padding="10")
        info_frame.grid(row=0, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 15))
        info_frame.columnconfigure(1, weight=1)

        ttk.Label(info_frame, text="Файл:", font=(self.default_font, self.font_size, 'bold')).grid(row=0, column=0, sticky=tk.W,
padx=5)
        self.file_label = ttk.Label(info_frame, text="Не выбран", foreground="gray", font=(self.default_font, self.font_size))
        self.file_label.grid(row=0, column=1, sticky=tk.W, padx=(5, 20))

        ttk.Label(info_frame, text="Статус:", font=(self.default_font, self.font_size, 'bold')).grid(row=0, column=2, sticky=tk.W,
padx=(20, 0))
        self.status_label = ttk.Label(info_frame, text="Готов к работе", foreground="green", font=(self.default_font, self.font_size,
'bold'))
        self.status_label.grid(row=0, column=3, sticky=tk.W, padx=5)

        button_frame = ttk.Frame(info_frame)
        button_frame.grid(row=0, column=4, padx=(50, 0))

        ttk.Button(button_frame, text="Последовательность лексем", command=self.show_full_sequence).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Программа с лексемами", command=self.show_lexeme_program).pack(side=tk.LEFT, padx=5)

        left_frame = ttk.LabelFrame(main_frame, text="Исходный код на R", padding="10")
        left_frame.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S), padx=(0, 10))
        left_frame.columnconfigure(0, weight=1)
        left_frame.rowconfigure(0, weight=1)

        self.code_text = scrolledtext.ScrolledText(
            left_frame,
            wrap=tk.WORD,
            font=(self.default_font, self.font_size),
            background='#ffffff',
            foreground='#000000',
            insertbackground='black'
        )
        self.code_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self._setup_code_tags()

        control_frame = ttk.Frame(left_frame)
        control_frame.grid(row=1, column=0, pady=15)

        ttk.Button(control_frame, text="Открыть файл", command=self.open_file, width=20).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Запустить анализ", command=self.analyze, width=20).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Очистить", command=self.clear_code, width=20).pack(side=tk.LEFT, padx=5)

        right_frame = ttk.LabelFrame(main_frame, text="Результаты анализа", padding="10")
        right_frame.grid(row=1, column=1, sticky=(tk.W, tk.E, tk.N, tk.S), padx=(10, 0))
        right_frame.columnconfigure(0, weight=1)
        right_frame.rowconfigure(0, weight=1)

        self.notebook = ttk.Notebook(right_frame)
        self.notebook.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self._setup_notebook_tabs()

        stats_frame = ttk.LabelFrame(main_frame, text="Статистика", padding="10")
        stats_frame.grid(row=2, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(15, 0))
        stats_frame.columnconfigure(1, weight=1)
        stats_frame.columnconfigure(3, weight=1)
        stats_frame.columnconfigure(5, weight=1)
        stats_frame.columnconfigure(7, weight=1)

        self.stats_labels = {}
        stats_items = [
            ("Всего лексем:", "0", 0),
            ("Идентификаторов:", "0", 2),
            ("Чисел:", "0", 4),
            ("Строк:", "0", 6),
            ("Комментариев:", "0", 8),
            ("Ключевых слов:", "0", 10),
            ("Операций:", "0", 12),
            ("Ошибок:", "0", 14)
        ]

        for i, (label, value, col) in enumerate(stats_items):
            ttk.Label(stats_frame, text=label, font=(self.default_font, self.font_size, 'bold')).grid(row=0, column=col, sticky=tk.W,
padx=(20 if i > 0 else 0, 5))
            self.stats_labels[label] = ttk.Label(stats_frame, text=value, foreground="blue", font=(self.default_font, self.font_size,
'bold'))
            self.stats_labels[label].grid(row=0, column=col + 1, sticky=tk.W, padx=(5, 20))
```

```
        self.progress = ttk.Progressbar(stats_frame, mode='indeterminate', length=250)
        self.progress.grid(row=0, column=16, padx=(50, 0))
        self.progress.grid_remove()

    def _setup_code_tags(self):
        self.code_text.tag_configure("keyword", foreground="#0000ff", font=(self.default_font, self.font_size, 'bold'))
        self.code_text.tag_configure("string", foreground="#008000", font=(self.default_font, self.font_size))
        self.code_text.tag_configure("comment", foreground="#808080", font=(self.default_font, self.font_size, 'italic'))
        self.code_text.tag_configure("number", foreground="#ff8c00", font=(self.default_font, self.font_size, 'bold'))
        self.code_text.tag_configure("operation", foreground="#ff00ff", font=(self.default_font, self.font_size, 'bold'))
        self.code_text.tag_configure("error", foreground="#ff0000", background="#fff0f0", font=(self.default_font, self.font_size,
'bold'))

    def _setup_notebook_tabs(self):
        self.tokens_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.tokens_frame, text="Лексемы")
        self.tokens_frame.columnconfigure(0, weight=1)
        self.tokens_frame.rowconfigure(0, weight=1)

        self.tokens_text = scrolledtext.ScrolledText(
            self.tokens_frame,
            wrap=tk.WORD,
            font=(self.default_font, self.small_font_size),
            background='#ffffff'
        )
        self.tokens_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.tables_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.tables_frame, text="Таблицы лексем")
        self.tables_frame.columnconfigure(0, weight=1)
        self.tables_frame.rowconfigure(0, weight=1)

        self.tables_text = scrolledtext.ScrolledText(
            self.tables_frame,
            wrap=tk.WORD,
            font=(self.default_font, self.small_font_size),
            background='#ffffff'
        )
        self.tables_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.errors_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.errors_frame, text="Ошибки")
        self.errors_frame.columnconfigure(0, weight=1)
        self.errors_frame.rowconfigure(0, weight=1)

        self.errors_text = scrolledtext.ScrolledText(
            self.errors_frame,
            wrap=tk.WORD,
            font=(self.default_font, self.small_font_size),
            background='#fff0f0',
            foreground='#ff0000'
        )
        self.errors_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.identifiers_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.identifiers_frame, text="Идентификаторы")
        self.identifiers_frame.columnconfigure(0, weight=1)
        self.identifiers_frame.rowconfigure(0, weight=1)

        self.identifiers_tree = ttk.Treeview(
            self.identifiers_frame,
            columns=('ID', 'Имя'),
            show='headings',
            height=15
        )
        self.identifiers_tree.heading('ID', text='ID')
        self.identifiers_tree.heading('Имя', text='Имя')
        self.identifiers_tree.column('ID', width=120, minwidth=80)
        self.identifiers_tree.column('Имя', width=350, minwidth=200)

        scrollbar_id = ttk.Scrollbar(self.identifiers_frame, orient=tk.VERTICAL, command=self.identifiers_tree.yview)
        self.identifiers_tree.configure(yscrollcommand=scrollbar_id.set)

        self.identifiers_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        scrollbar_id.grid(row=0, column=1, sticky=(tk.N, tk.S))

        self.numbers_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.numbers_frame, text="Числа")
        self.numbers_frame.columnconfigure(0, weight=1)
        self.numbers_frame.rowconfigure(0, weight=1)

        self.numbers_tree = ttk.Treeview(
            self.numbers_frame,
            columns=('ID', 'Значение', 'Тип', 'Статус'),
            show='headings',
            height=15
        )
        self.numbers_tree.heading('ID', text='ID')
        self.numbers_tree.heading('Значение', text='Значение')
        self.numbers_tree.heading('Тип', text='Тип')
```

```
        self.numbers_tree.heading('Статус', text='Статус')
        self.numbers_tree.column('ID', width=120, minwidth=80)
        self.numbers_tree.column('Значение', width=180, minwidth=120)
        self.numbers_tree.column('Тип', width=180, minwidth=120)
        self.numbers_tree.column('Статус', width=250, minwidth=180)

        scrollbar_num = ttk.Scrollbar(self.numbers_frame, orient=tk.VERTICAL, command=self.numbers_tree.yview)
        self.numbers_tree.configure(yscrollcommand=scrollbar_num.set)

        self.numbers_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        scrollbar_num.grid(row=0, column=1, sticky=(tk.N, tk.S))

        self.comments_frame = ttk.Frame(self.notebook)
        self.notebook.add(self.comments_frame, text="Комментарии")
        self.comments_frame.columnconfigure(0, weight=1)
        self.comments_frame.rowconfigure(0, weight=1)

        self.comments_tree = ttk.Treeview(
            self.comments_frame,
            columns=('ID', 'Комментарий'),
            show='headings',
            height=15
        )
        self.comments_tree.heading('ID', text='ID')
        self.comments_tree.heading('Комментарий', text='Комментарий')
        self.comments_tree.column('ID', width=120, minwidth=80)
        self.comments_tree.column('Комментарий', width=600, minwidth=400)

        scrollbar_com = ttk.Scrollbar(self.comments_frame, orient=tk.VERTICAL, command=self.comments_tree.yview)
        self.comments_tree.configure(yscrollcommand=scrollbar_com.set)

        self.comments_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        scrollbar_com.grid(row=0, column=1, sticky=(tk.N, tk.S))

    def load_example(self, example_type):
        examples = {
            "correct": """# Пример корректного кода на R
calculate_stats <- function(data) {
    mean_val <- mean(data, na.rm = TRUE)
    sd_val <- sd(data, na.rm = TRUE)

    if (mean_val > 0) {
        result <- list(
            mean = mean_val,
            sd = sd_val,
            n = length(data)
        )
        return(result)
    } else {
        return(NULL)
    }
}

x <- 123.45
y <- 2.5e-3
z <- 100
w <- 1.6E-19""",

            "correct_numbers": """# Примеры КОРРЕКТНЫХ чисел в R

a <- 42
b <- 0
c <- 1000000
d <- -123

e <- 123.45
g <- 0.5
h <- -3.14
i <- 8.05

j <- 2.5e-3
k <- 1.6E-19
l <- 3e5
n <- 123.e-4

x <- c(1, 2, 3)
y <- list$element
z <- 1.2""",

            "errors": """# Примеры НЕКОРРЕКТНЫХ чисел в R

a <- 123.23.3
b <- 1.2.3.4
c <- 123..45

d <- 123a
e <- 123b213a
f <- 45x67
g <- 1a2a3
h <- 1a2b3c
```

```
i ← 2.5e
j ← 1.5e-
k ← 3e2.5
l ← 4e2a""",

            "dots": """# Примеры использования точки в R

price ← 123.23.3
version ← 1.2.3.4
value ← 123...45
coord ← 12.34.56.78

correct1 ← 123.45
correct2 ← 5.07
correct3 ← 10.2
correct4 ← 2.5e-3

x ← list(a=1, b=2)
y ← x$a
z ← 1.2""",

            "letters": """# Примеры букв в числах

a ← 123a
b ← 45x
c ← 67y89
d ← 123b213a
e ← 45x67y89
f ← 1a2a3
g ← 1a2b3c

h ← 2.5ea
i ← 1.3e2b

j ← 123
k ← 2.5e-3
l ← 1.6E-19
m ← 0.5

var123 ← 10
x2 ← 20
test_a ← 30"""
        }

        if example_type in examples:
            self.code_text.delete(1.0, tk.END)
            self.code_text.insert(1.0, examples[example_type])
            self.current_file = None

            self._clear_code_tags()

            name_map = {
                "correct": "Пример: корректный код",
                "correct_numbers": "Пример: корректные числа",
                "errors": "Пример: ошибки в числах",
                "dots": "Пример: использование точки",
                "letters": "Пример: буквы в числах"
            }
            self.file_label.config(text=name_map.get(example_type, "Пример"), foreground="green")

    def _clear_code_tags(self):
        for tag in ["keyword", "string", "comment", "number", "operation", "error"]:
            self.code_text.tag_remove(tag, 1.0, tk.END)

    def open_file(self):
        filename = filedialog.askopenfilename(
            title="Выберите файл с кодом R",
            filetypes=[("R files", "*.r"), ("R files", "*.R"), ("Text files", "*.txt"), ("All files", "*.*")]
        )

        if filename:
            try:
                with open(filename, 'r', encoding='utf-8') as f:
                    content = f.read()

                self.code_text.delete(1.0, tk.END)
                self.code_text.insert(1.0, content)
                self.current_file = filename
                self.file_label.config(text=Path(filename).name, foreground="black")
                self.status_label.config(text="Файл загружен", foreground="green")

                self._clear_code_tags()

            except Exception as e:
                messagebox.showerror("Ошибка", f"Не удалось открыть файл:\n{str(e)}")

    def highlight_syntax(self):
        self._clear_code_tags()
```

```
    content = self.code_text.get(1.0, tk.END)

    for match in re.finditer(r'#.*$', content, re.MULTILINE):
        start = f"1.0 + {match.start()} chars"
        end = f"1.0 + {match.end()} chars"
        self.code_text.tag_add("comment", start, end)

    for match in re.finditer(r'"[^"\\]*(\\.[^"\\]*)*"|\'[^\'\\]*(\\.[^\'\\]*)*\'', content):
        start = f"1.0 + {match.start()} chars"
        end = f"1.0 + {match.end()} chars"
        self.code_text.tag_add("string", start, end)

    for kw in self.lexer.keywords.keys():
        start = 1.0
        while True:
            start = self.code_text.search(r'\m' + re.escape(kw) + r'\M', start, tk.END)
            if not start:
                break
            end = f"{start}+{len(kw)}c"
            self.code_text.tag_add("keyword", start, end)
            start = end

    for match in re.finditer(r'\b\d+\.?\d*(?:[eE][+-]?\d+)?\b|\b\.\d+(?:[eE][+-]?\d+)?\b|\b\d+\.\b', content):
        start = f"1.0 + {match.start()} chars"
        end = f"1.0 + {match.end()} chars"
        self.code_text.tag_add("number", start, end)

def highlight_errors(self):
    self.code_text.tag_remove("error", 1.0, tk.END)

    for token in self.lexer.token_sequence:
        if token.lex_type == LexemType.ERROR:
            start = f"{token.line}.0 + {token.column - 1} chars"
            end = f"{start} + {len(token.value)} chars"
            self.code_text.tag_add("error", start, end)

def analyze(self):
    code = self.code_text.get(1.0, tk.END)

    if not code.strip():
        messagebox.showwarning("Предупреждение", "Нет кода для анализа!")
        return

    self.progress.grid()
    self.progress.start()
    self.status_label.config(text="Выполняется анализ...", foreground="orange")
    self.root.update()

    try:
        tokens = self.lexer.tokenize(code)

        self.update_results()

        self.update_statistics()

        self.highlight_errors()

        if self.lexer.errors:
            self.status_label.config(text=f"Анализ завершен. Найдено ошибок: {len(self.lexer.errors)}", foreground="red")
            self.notebook.select(self.errors_frame)
        else:
            self.status_label.config(text="Анализ завершен. Ошибок не обнаружено", foreground="green")

    except Exception as e:
        self.status_label.config(text="Ошибка анализа", foreground="red")
        messagebox.showerror("Ошибка", f"Ошибка при анализе:\n{str(e)}")
        import traceback
        traceback.print_exc()

    finally:
        self.progress.stop()
        self.progress.grid_remove()

def show_full_sequence(self):
    if not self.lexer.token_sequence:
        messagebox.showwarning("Предупреждение", "Нет данных для отображения! Сначала выполните анализ.")
        return

    full_window = tk.Toplevel(self.root)
    full_window.title("Полная последовательность лексем")
    full_window.geometry("1200x700")
    full_window.configure(bg='#f0f0f0')
    full_window.option_add('*Font', (self.default_font, self.small_font_size))

    main_frame = ttk.Frame(full_window, padding="15")
    main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
    full_window.columnconfigure(0, weight=1)
    full_window.rowconfigure(0, weight=1)
    main_frame.columnconfigure(0, weight=1)
    main_frame.rowconfigure(1, weight=1)
```

```
        title_label = ttk.Label(main_frame, text="ПОЛНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ЛЕКСЕМ",
                                font=(self.default_font, self.large_font_size, 'bold'))
        title_label.grid(row=0, column=0, pady=(0, 15))

        text_frame = ttk.Frame(main_frame)
        text_frame.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        text_frame.columnconfigure(0, weight=1)
        text_frame.rowconfigure(0, weight=1)

        full_text = scrolledtext.ScrolledText(
            text_frame,
            wrap=tk.WORD,
            font=(self.default_font, self.small_font_size),
            background='#ffffff'
        )
        full_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self._insert_full_sequence_content(full_text)

        ttk.Button(main_frame, text="Закрыть", command=full_window.destroy, width=15).grid(row=2, column=0, pady=15)

    def _insert_full_sequence_content(self, text_widget):
        text_widget.insert(1.0, "=" * 100 + "\n")
        text_widget.insert(2.0, f"ПОЛНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ЛЕКСЕМ ({len(self.lexer.token_sequence)} шт.)\n")
        text_widget.insert(3.0, "=" * 100 + "\n\n")

        current_line = 1
        line_tokens = []

        for token in self.lexer.token_sequence:
            if token.line > current_line:
                if line_tokens:
                    text_widget.insert(tk.END, f"\n--- Строка {current_line} ---\n")
                    for i, t in enumerate(line_tokens, 1):
                        status = "ОШИБКА" if t.lex_type == LexemType.ERROR else "ОК"
                        text_widget.insert(tk.END, f"{i:4d}. {t.code:8s} | {t.value:25s} | позиция {t.column:4d} | {status}\n")
                        if t.error_msg:
                            text_widget.insert(tk.END, f"      └ {t.error_msg}\n")
                line_tokens = []
                current_line = token.line
            line_tokens.append(token)

        if line_tokens:
            text_widget.insert(tk.END, f"\n--- Строка {current_line} ---\n")
            for i, t in enumerate(line_tokens, 1):
                status = "ОШИБКА" if t.lex_type == LexemType.ERROR else "ОК"
                text_widget.insert(tk.END, f"{i:4d}. {t.code:8s} | {t.value:25s} | позиция {t.column:4d} | {status}\n")
                if t.error_msg:
                    text_widget.insert(tk.END, f"      └ {t.error_msg}\n")

        text_widget.insert(tk.END, "\n" + "=" * 100 + "\n")
        text_widget.insert(tk.END, "СТАТИСТИКА ПО ЛЕКСЕМАМ\n")
        text_widget.insert(tk.END, "=" * 100 + "\n")

        type_counts = {}
        for token in self.lexer.token_sequence:
            type_name = {
                LexemType.KEYWORD: "Служебные слова",
                LexemType.IDENTIFIER: "Идентификаторы",
                LexemType.NUMBER: "Числа",
                LexemType.STRING: "Строки",
                LexemType.OPERATION: "Операции",
                LexemType.DELIMITER: "Разделители",
                LexemType.COMMENT: "Комментарии",
                LexemType.ERROR: "Ошибки"
            }.get(token.lex_type, token.lex_type.value)

            type_counts[type_name] = type_counts.get(type_name, 0) + 1

        for type_name, count in sorted(type_counts.items()):
            text_widget.insert(tk.END, f"{type_name:20s}: {count:4d} лексем\n")

    def show_lexeme_program(self):
        if not self.lexer.token_sequence:
            messagebox.showwarning("Предупреждение", "Нет данных для отображения! Сначала выполните анализ.")
            return

        lex_window = tk.Toplevel(self.root)
        lex_window.title("Программа с лексемами")
        lex_window.geometry("1200x800")
        lex_window.configure(bg='#f0f0f0')
        lex_window.option_add('*Font', (self.default_font, self.small_font_size))

        main_frame = ttk.Frame(lex_window, padding="15")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        lex_window.columnconfigure(0, weight=1)
        lex_window.rowconfigure(0, weight=1)
        main_frame.columnconfigure(0, weight=1)
        main_frame.rowconfigure(1, weight=1)
```

```
title_label = ttk.Label(main_frame, text="ПРОГРАММА С ЗАМЕНОЙ НА ЛЕКСЕМЫ",
                        font=(self.default_font, self.large_font_size, 'bold'))
title_label.grid(row=0, column=0, pady=(0, 15))

paned = ttk.PanedWindow(main_frame, orient=tk.VERTICAL)
paned.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

top_frame = ttk.Frame(paned)
paned.add(top_frame, weight=2)
top_frame.columnconfigure(0, weight=1)
top_frame.rowconfigure(1, weight=1)

ttk.Label(top_frame, text="Исходный код с заменой на лексемы:",
          font=(self.default_font, self.font_size, 'bold')).grid(row=0, column=0, sticky=tk.W, pady=(0, 10))

lex_text = scrolledtext.ScrolledText(
    top_frame,
    wrap=tk.WORD,
    font=(self.default_font, self.font_size),
    background='#ffffff'
)
lex_text.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

clean_lex_program = self.lexer.generate_clean_lexeme_program()
lex_text.insert(1.0, clean_lex_program)

bottom_frame = ttk.Frame(paned)
paned.add(bottom_frame, weight=1)
bottom_frame.columnconfigure(0, weight=1)
bottom_frame.rowconfigure(1, weight=1)

ttk.Label(bottom_frame, text="Соответствие лексем:",
          font=(self.default_font, self.font_size, 'bold')).grid(row=0, column=0, sticky=tk.W, pady=(0, 10))

bottom_notebook = ttk.Notebook(bottom_frame)
bottom_notebook.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

kw_frame = ttk.Frame(bottom_notebook)
bottom_notebook.add(kw_frame, text="Ключевые слова")
kw_frame.columnconfigure(0, weight=1)
kw_frame.rowconfigure(0, weight=1)

kw_text = scrolledtext.ScrolledText(
    kw_frame,
    wrap=tk.WORD,
    font=(self.default_font, self.small_font_size),
    background='#ffffff'
)
kw_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

for word, idx in sorted(self.lexer.keywords.items(), key=lambda x: x[1]):
    kw_text.insert(tk.END, f"W{idx:4d} : {word}\n")

id_frame = ttk.Frame(bottom_notebook)
bottom_notebook.add(id_frame, text="Идентификаторы")
id_frame.columnconfigure(0, weight=1)
id_frame.rowconfigure(0, weight=1)

id_text = scrolledtext.ScrolledText(
    id_frame,
    wrap=tk.WORD,
    font=(self.default_font, self.small_font_size),
    background='#ffffff'
)
id_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

for name, idx in sorted(self.lexer.identifiers.items(), key=lambda x: x[1]):
    id_text.insert(tk.END, f"I{idx:4d} : {name}\n")

num_frame = ttk.Frame(bottom_notebook)
bottom_notebook.add(num_frame, text="Числа")
num_frame.columnconfigure(0, weight=1)
num_frame.rowconfigure(0, weight=1)

num_text = scrolledtext.ScrolledText(
    num_frame,
    wrap=tk.WORD,
    font=(self.default_font, self.small_font_size),
    background='#ffffff'
)
num_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

for num, idx in sorted(self.lexer.numbers.items(), key=lambda x: x[1]):
    num_type = self.classify_number(num)
    num_text.insert(tk.END, f"N{idx:4d} : {num:15s} - {num_type}\n")

com_frame = ttk.Frame(bottom_notebook)
bottom_notebook.add(com_frame, text="Комментарии")
com_frame.columnconfigure(0, weight=1)
```

```
    com_frame.rowconfigure(0, weight=1)

    com_text = scrolledtext.ScrolledText(
        com_frame,
        wrap=tk.WORD,
        font=(self.default_font, self.small_font_size),
        background='#ffffff'
    )
    com_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

    for comment, idx in sorted(self.lexer.comments.items(), key=lambda x: x[1]):
        com_text.insert(tk.END, f"C{idx:4d} : {comment}\n")

    lex_text.config(state=tk.DISABLED)
    for widget in [kw_text, id_text, num_text, com_text]:
        widget.config(state=tk.DISABLED)

    button_frame = ttk.Frame(main_frame)
    button_frame.grid(row=2, column=0, pady=15)

    ttk.Button(button_frame, text="Сохранить в файл",
               command=lambda: self.save_lexeme_program(clean_lex_program), width=20).pack(side=tk.LEFT, padx=10)
    ttk.Button(button_frame, text="Закрыть",
               command=lex_window.destroy, width=15).pack(side=tk.LEFT, padx=10)

def save_lexeme_program(self, content):
    filename = filedialog.asksaveasfilename(
        title="Сохранить программу с лексемами",
        defaultextension=".txt",
        filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
    )

    if filename:
        try:
            with open(filename, 'w', encoding='utf-8') as f:
                f.write(content)
            messagebox.showinfo("Успех", f"Программа сохранена в файл:\n{filename}")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось сохранить файл:\n{str(e)}")

def update_results(self):
    self._update_tokens_text()
    self._update_tables_text()
    self._update_errors_text()
    self._update_trees()

def _update_tokens_text(self):
    self.tokens_text.delete(1.0, tk.END)
    self.tokens_text.insert(1.0, "ПОСЛЕДОВАТЕЛЬНОСТЬ ЛЕКСЕМ (первые 100)\n")
    self.tokens_text.insert(2.0, "=" * 90 + "\n")
    self.tokens_text.insert(3.0, f"{'№':4s} {'Код':8s} {'Значение':25s} {'Строка':8s} {'Колонка':8s} {'Статус':15s}\n")
    self.tokens_text.insert(4.0, "-" * 90 + "\n")

    for i, token in enumerate(self.lexer.token_sequence[:100], 1):
        status = "ОШИБКА" if token.lex_type == LexemType.ERROR else "ОК"
        line = f"{i:4d} {token.code:8s} {token.value:25s} {token.line:8d} {token.column:8d} {status:15s}\n"
        self.tokens_text.insert(tk.END, line)
        if token.error_msg:
            self.tokens_text.insert(tk.END, f"      └─ {token.error_msg}\n")

    if len(self.lexer.token_sequence) > 100:
        self.tokens_text.insert(tk.END, f"\n... и еще {len(self.lexer.token_sequence) - 100} лексем. ")
        self.tokens_text.insert(tk.END, "Используйте меню 'Просмотр > Полная последовательность лексем' для просмотра всех.\n")

def _update_tables_text(self):
    self.tables_text.delete(1.0, tk.END)
    self.tables_text.insert(1.0, self.format_tables())

def _update_errors_text(self):
    self.errors_text.delete(1.0, tk.END)
    if self.lexer.errors:
        self.errors_text.insert(1.0, "=" * 80 + "\n")
        self.errors_text.insert(2.0, "НАЙДЕНЫ ОШИБКИ ЛЕКСИЧЕСКОГО АНАЛИЗА\n")
        self.errors_text.insert(3.0, "=" * 80 + "\n\n")

        dot_errors = [e for e in self.lexer.errors if "точк" in e.lower()]
        letter_errors = [e for e in self.lexer.errors if "букв" in e.lower()]
        other_errors = [e for e in self.lexer.errors if e not in dot_errors and e not in letter_errors]

        if dot_errors:
            self.errors_text.insert(tk.END, "  НЕКОРРЕКТНОЕ ИСПОЛЬЗОВАНИЕ ТОЧКИ:\n")
            self.errors_text.insert(tk.END, "-" * 40 + "\n")
            for i, error in enumerate(dot_errors, 1):
                self.errors_text.insert(tk.END, f"{i}. {error}\n")
            self.errors_text.insert(tk.END, "\n")

        if letter_errors:
            self.errors_text.insert(tk.END, "  БУКВЫ В ЧИСЛАХ:\n")
            self.errors_text.insert(tk.END, "-" * 40 + "\n")
            for i, error in enumerate(letter_errors, 1):
```

```
                self.errors_text.insert(tk.END, f"{i}. {error}\n")
            self.errors_text.insert(tk.END, "\n")

        if other_errors:
            self.errors_text.insert(tk.END, "  ПРОЧИЕ ОШИБКИ:\n")
            self.errors_text.insert(tk.END, "-" * 40 + "\n")
            for i, error in enumerate(other_errors, 1):
                self.errors_text.insert(tk.END, f"{i}. {error}\n")
    else:
        self.errors_text.insert(1.0, "  Ошибок не обнаружено.\n")

def _update_trees(self):
    for item in self.identifiers_tree.get_children():
        self.identifiers_tree.delete(item)

    for name, idx in sorted(self.lexer.identifiers.items(), key=lambda x: x[1]):
        self.identifiers_tree.insert('', tk.END, values=(f"I{idx}", name))

    for item in self.numbers_tree.get_children():
        self.numbers_tree.delete(item)

    for num, idx in sorted(self.lexer.numbers.items(), key=lambda x: x[1]):
        num_type = self.classify_number(num)
        self.numbers_tree.insert('', tk.END, values=(f"N{idx}", num, num_type, "Корректное"))

    error_numbers = set()
    for token in self.lexer.token_sequence:
        if token.lex_type == LexemType.ERROR and any(c.isdigit() for c in token.value):
            if token.value not in error_numbers:
                error_numbers.add(token.value)
                if '.' in token.value:
                    if 'e' in token.value.lower():
                        num_type = "число с плавающей точкой"
                    else:
                        num_type = "число с фиксированной точкой"
                else:
                    num_type = "целое число"
                self.numbers_tree.insert('', tk.END, values=("E", token.value, num_type, f"ОШИБКА: {token.error_msg}"))

    for item in self.comments_tree.get_children():
        self.comments_tree.delete(item)

    for comment, idx in sorted(self.lexer.comments.items(), key=lambda x: x[1]):
        self.comments_tree.insert('', tk.END, values=(f"C{idx}", comment))

def format_tables(self):
    output = []

    output.append("=" * 80)
    output.append("ТАБЛИЦЫ ЛЕКСЕМ")
    output.append("=" * 80)

    output.append("\n1. СЛУЖЕБНЫЕ СЛОВА:")
    output.append("-" * 40)
    for word, idx in sorted(self.lexer.keywords.items(), key=lambda x: x[1]):
        output.append(f"  W{idx:4d} : {word}")

    output.append("\n2. РАЗДЕЛИТЕЛИ:")
    output.append("-" * 40)
    for delim, idx in sorted(self.lexer.delimiters.items(), key=lambda x: x[1]):
        repr_delim = repr(delim).strip("'")
        output.append(f"  R{idx:4d} : {repr_delim}")

    output.append("\n3. ОПЕРАЦИИ:")
    output.append("-" * 40)
    for op, idx in sorted(self.lexer.operations.items(), key=lambda x: x[1]):
        output.append(f"  O{idx:4d} : {op}")

    output.append("\n4. ИДЕНТИФИКАТОРЫ:")
    output.append("-" * 40)
    if self.lexer.identifiers:
        for name, idx in sorted(self.lexer.identifiers.items(), key=lambda x: x[1]):
            output.append(f"  I{idx:4d} : {name}")
    else:
        output.append("  Нет идентификаторов")

    output.append("\n5. ЧИСЛА:")
    output.append("-" * 40)
    if self.lexer.numbers:
        for num, idx in sorted(self.lexer.numbers.items(), key=lambda x: x[1]):
            num_type = self.classify_number(num)
            output.append(f"  N{idx:4d} : {num:15s} - {num_type}")
    else:
        output.append("  Нет чисел")

    output.append("\n6. СТРОКИ:")
    output.append("-" * 40)
    if self.lexer.strings:
        for string, idx in sorted(self.lexer.strings.items(), key=lambda x: x[1]):
            output.append(f"  S{idx:4d} : {string}")
```

```
            else:
                output.append("  Нет строк")

            output.append("\n7. КОММЕНТАРИИ:")
            output.append("-" * 40)
            if self.lexer.comments:
                for comment, idx in sorted(self.lexer.comments.items(), key=lambda x: x[1]):
                    output.append(f"  C{idx:4d} : {comment}")
            else:
                output.append("  Нет комментариев")

            output.append("\n8. СТАТИСТИКА ОШИБОК:")
            output.append("-" * 40)
            output.append(f"  Всего ошибок: {len(self.lexer.errors)}")

            dot_errors = len([e for e in self.lexer.errors if "точк" in e.lower()])
            letter_errors = len([e for e in self.lexer.errors if "букв" in e.lower()])
            other_errors = len(self.lexer.errors) - dot_errors - letter_errors

            output.append(f"  - Некорректное использование точки: {dot_errors}")
            output.append(f"  - Буквы в числах: {letter_errors}")
            output.append(f"  - Прочие ошибки: {other_errors}")

            return '\n'.join(output)

        def classify_number(self, num_str):
            if '.' in num_str:
                if 'e' in num_str.lower():
                    return "число с плавающей точкой"
                else:
                    return "число с фиксированной точкой"
            else:
                return "целое число"

        def update_statistics(self):
            error_count = len([t for t in self.lexer.token_sequence if t.lex_type == LexemType.ERROR])

            stats = {
                "Всего лексем:": len(self.lexer.token_sequence),
                "Идентификаторов:": len(self.lexer.identifiers),
                "Чисел:": len(self.lexer.numbers),
                "Строк:": len(self.lexer.strings),
                "Комментариев:": len(self.lexer.comments),
                "Ключевых слов:": len([t for t in self.lexer.token_sequence if t.lex_type == LexemType.KEYWORD]),
                "Операций:": len([t for t in self.lexer.token_sequence if t.lex_type == LexemType.OPERATION]),
                "Ошибок:": error_count
            }

            for label, value in stats.items():
                if label in self.stats_labels:
                    color = "red" if label == "Ошибок:" and value > 0 else "blue"
                    self.stats_labels[label].config(text=str(value), foreground=color)

        def save_results(self):
            if not self.lexer.token_sequence:
                messagebox.showwarning("Предупреждение", "Нет результатов для сохранения!")
                return

            filename = filedialog.asksaveasfilename(
                title="Сохранить результаты",
                defaultextension=".txt",
                filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
            )

            if filename:
                try:
                    with open(filename, 'w', encoding='utf-8') as f:
                        f.write("РЕЗУЛЬТАТЫ ЛЕКСИЧЕСКОГО АНАЛИЗА\n")
                        f.write("=" * 80 + "\n\n")
                        f.write(f"Дата анализа: {datetime.datetime.now()}\n")
                        f.write(f"Исходный файл: {self.current_file or 'Ввод с редактора'}\n\n")

                        f.write("ПОСЛЕДОВАТЕЛЬНОСТЬ ЛЕКСЕМ:\n")
                        f.write("-" * 90 + "\n")
                        f.write(f"{'№':6s} {'Код':8s} {'Значение':25s} {'Строка':8s} {'Колонка':8s} {'Статус':15s}\n")
                        f.write("-" * 90 + "\n")

                        for i, token in enumerate(self.lexer.token_sequence, 1):
                            status = "ОШИБКА" if token.lex_type == LexemType.ERROR else "ОК"
                            f.write(f"{i:6d} | {token.code:8s} | {token.value:25s} | {token.line:8d} | {token.column:8d} |
{status:15s}\n")
                            if token.error_msg:
                                f.write(f"      └─ {token.error_msg}\n")

                        f.write("\n\n" + self.format_tables())

                        f.write("\n\n" + "=" * 80 + "\n")
                        f.write("ПРОГРАММА С ЗАМЕНОЙ НА ЛЕКСЕМЫ\n")
                        f.write("=" * 80 + "\n\n")
                        f.write(self.lexer.generate_lexeme_program())
```

```
                messagebox.showinfo("Успех", f"Результаты сохранены в файл:\n{filename}")

            except Exception as e:
                messagebox.showerror("Ошибка", f"Не удалось сохранить файл:\n{str(e)}")

    def clear_code(self):
        self.code_text.delete(1.0, tk.END)
        self.current_file = None
        self.file_label.config(text="Не выбран", foreground="gray")
        self._clear_code_tags()

    def clear_all(self):
        self.clear_code()
        self.tokens_text.delete(1.0, tk.END)
        self.tables_text.delete(1.0, tk.END)
        self.errors_text.delete(1.0, tk.END)

        for item in self.identifiers_tree.get_children():
            self.identifiers_tree.delete(item)

        for item in self.numbers_tree.get_children():
            self.numbers_tree.delete(item)

        for item in self.comments_tree.get_children():
            self.comments_tree.delete(item)

        for label in self.stats_labels:
            self.stats_labels[label].config(text="0", foreground="blue")

        self.lexer.reset()
        self.status_label.config(text="Готов к работе", foreground="green")

    def show_about(self):
        about_text = f"""Краснодар 2026"""
        HelpWindow(self.root, "О программе", about_text, width=600, height=400)

    def show_r_syntax(self):
        syntax_text = """СИНТАКСИС ЯЗЫКА R

Ключевые слова:
if, else, while, for, function, return, TRUE, FALSE, NULL, NA, Inf, NaN

Операторы присваивания:
=, ←, ↞, ->, ->>, %

Арифметические операторы:
+, -, *, /, ^, %%, %/%

Логические операторы:
&, |, !, &&, ||, xor()

Операторы сравнения:
<, >, ⩽, ⩾, ==, ≠

Специальные операторы:
%in%, %*%, %>%

Комментарии:
# Однострочные комментарии

Строки:
'в одинарных кавычках' или "в двойных кавычках"

Разделители:
, ; : :: ::: ( ) [ ] [[ ]] { } ` $ @"""
        HelpWindow(self.root, "Синтаксис R", syntax_text, width=700, height=500)

    def show_error_types(self):
        error_text = f"""ТИПЫ ОШИБОК, ОТЛАВЛИВАЕМЫХ АНАЛИЗАТОРОМ:

1. НЕКОРРЕКТНОЕ ИСПОЛЬЗОВАНИЕ ТОЧКИ:
   • Множественные точки: 123.23.3, 1.2.3.4, 12.34.56.78
   • Несколько точек подряд: 123..45, 1..2, 5..2, 1...7
   • Точка в экспоненте: 1.5e2.3, 2.5e-3.4

2. БУКВЫ В ЧИСЛАХ:
   • Буквы после цифр: 123a, 45x, 67y
   • Буквы внутри числа: 1a2a3, 123b213a, 45x67y89
   • Буквы в экспоненте: 2.5ea, 1.3e2b

3. НЕКОРРЕКТНОЕ ПОСТРОЕНИЕ ЧИСЛА:
   • Начинается с цифры, содержит буквы: 123abc
   • Смешанный формат: 123a456

4. НЕЗАКРЫТЫЕ СТРОКИ:
   • Отсутствует закрывающая кавычка

5. НЕИЗВЕСТНЫЕ СИМВОЛЫ:
   • Символы, не принадлежащие алфавиту языка"""
```

```
        HelpWindow(self.root, "Типы ошибок", error_text, width=700, height=600)

if __name__ == "__main__":
    main()
```

5 Вывод

Разработанный лексический анализатор успешно выполняет разбор исходного кода на языке R. Программа корректно распознает все основные типы лексем, включая ключевые слова, идентификаторы, числовые и строковые литералы, операции и разделители.

Анализатор обладает следующими преимуществами:

- Поддержка полного набора ключевых слов и функций языка R
- Корректная обработка чисел в различных форматах (целые, с фиксированной и плавающей точкой)
- Выявление типичных лексических ошибок
- Удобный графический интерфейс с подсветкой синтаксиса
- Возможность сохранения результатов анализа

Программа может быть использована в качестве компонента полноценного транслятора для языка R или как самостоятельный инструмент для анализа и отладки кода.