Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №1**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Построение лексического анализатора»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л. В. Гаранина

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

Краснодар

2025

**Содержание**

[1. Вариант задания 3](#_Toc191420803)

[2. Функции и таблицы лексического анализа 3](#_Toc191420804)

[3. Результаты экспериментов 6](#_Toc191420805)

[ПРИЛОЖЕНИЕ А Листинг программы 9](#_Toc191420806)

# 1. Вариант задания

Вариант задания включает в себя входной язык и выходной язык, представленные в таблице 1.

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 45 | Java | R |

Необходимо разработать лексический анализатор, который сможет корректно идентифицировать лексемы, составлять соответствующие таблицы и формировать внутреннее представление исходного текста.

Программа принимает на вход файл с текстом, написанным на входном языке программирования. В результате работы должен быть сформирован файл с последовательностью кодов обнаруженных лексем, а также отдельные файлы с таблицами всех найденных лексем.

# 2. Функции и таблицы лексического анализа

Основная задача лексического анализатора — выделение лексем из исходного кода программы и передача их синтаксическому анализатору в определённом внутреннем представлении.

Для работы сканер использует таблицы, соответствующие различным классам лексем. Таблицы служебных слов, разделителей и операторов являются неизменяемыми и формируются на этапе разработки анализатора, так как они определяются входным языком программирования. В то же время таблицы идентификаторов и констант создаются динамически в процессе лексического анализа исходного кода.

Лексический анализатор пропускает комментарии, так как они служат для пояснения кода и не изменяют его работу.

Для языка Java таблицы имеют вид:

Таблица 2 – Таблица служебных слов

|  |  |
| --- | --- |
| Служебное слово | Код |
| if | 1 |
| else | 2 |
| while | 3 |
| for | 4 |
| return | 5 |
| String | 6 |
| int | 7 |
| float | 8 |
| double | 9 |
| boolean | 10 |
| class | 11 |
| public | 12 |
| private | 13 |
| static | 14 |
| void | 15 |
| char | 16 |
| new | 17 |
| this | 18 |
| true | 19 |
| false | 20 |
| null | 21 |
| break | 22 |
| continue | 23 |

Таблица 3 – Таблица операций

|  |  |
| --- | --- |
| Операция | Код |
| + | 1 |
| - | 2 |
| \* | 3 |
| / | 4 |
| = | 5 |
| == | 6 |
| < | 7 |
| > | 8 |
| <= | 9 |
| >= | 10 |
| != | 11 |
| && | 12 |
| || | 13 |
| ++ | 14 |
| -- | 15 |

Таблица 4 – Таблица разделителей

|  |  |
| --- | --- |
| Разделитель | Код |
| ; | 1 |
| , | 2 |
| { | 3 |
| } | 4 |
| ( | 5 |
| ) | 6 |
| [ | 7 |
| ] | 8 |
| . | 9 |
| @ | 10 |
| \n | 11 |
| пробел | 12 |

Каждая лексема, обработанная сканером, преобразуется к виду:

<буква><код>,

где: <буква> – это признак класса лексемы (W, I, O, R, N или C),   
<код> – номер лексемы в соответствующей таблице.

При разборе входного кода пробелы не учитываются как разделители и не сохраняются в выходных данных, так как их роль ограничивается структурированием текста без внесения смысловой информации.

# 3. Результаты экспериментов

На рисунке 1 представлена исходная программа на языке Java, на рисунке 2 – последовательность кодов лексем входной программы.

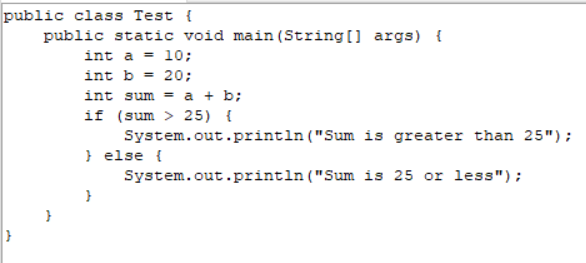


Рисунок 1 – Код исходной программы на Java

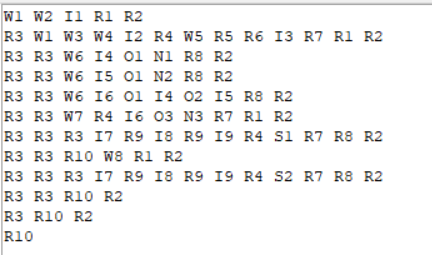
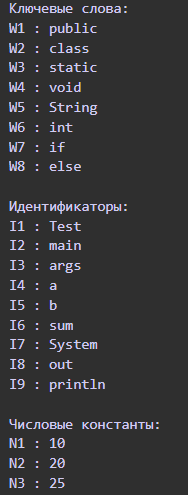


Рисунок 2 – Результат роботы программы

В результате работы программы создаются файлы result.txt и tables\_result.txt. На рисунках 3-4 показаны скриншоты файла tables\_result.txt, содержащего лексемы, присутствующие в коде исходной программе на Java.



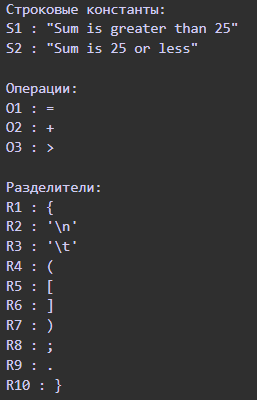


Рисунок 4 – 2 часть таблицы лексем

Рисунок 3 – 1 часть таблицы лексем

На рисунках 5-6 изображено содержимое файла result.txt. Это результат работы программы, сопоставляющий каждой лексеме программы определенную категорию и индекс.

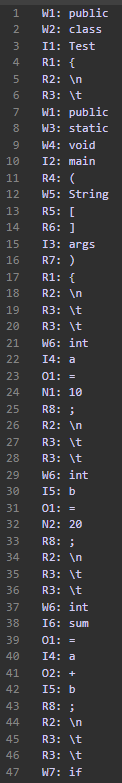


Рисунок 5 – 1 часть файла результатов

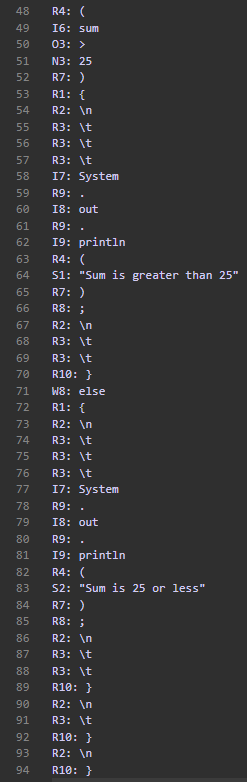


Рисунок 6 – 2 часть файла результатов

# ПРИЛОЖЕНИЕ А

**Листинг программы**

import tkinter as tk

from tkinter import filedialog, messagebox, ttk

import re

KEYWORDS\_JAVA = {"if", "else", "while", "for", "return", "String", "int", "float", "double", "boolean", "class", "public", "private", "static", "void", "char", "new", "this", "true", "false", "null", "break", "continue"}

OPERATORS\_JAVA = {"+", "-", "\*", "/", "=", "==", "<", ">", "<=", ">=", "!=", "&&", "||", "++", "--"}

DELIMITERS = {";", ",", "{", "}", "(", ")", "[", "]", ".", "@", "\n", "\t"}

TOKEN\_PATTERNS = [

(r"\b\d+(\.\d+)?\b", "Числовые константы"),

(r'".\*?"', "Строковые константы"),

(r"\b[a-zA-Z\_][a-zA0-9\_]\*\b", "Идентификаторы"),

]

tables = {

"Ключевые слова": {},

"Идентификаторы": {},

"Числовые константы": {},

"Строковые константы": {},

"Операции": {},

"Разделители": {}

}

category\_prefixes = {

"Ключевые слова": "W",

"Идентификаторы": "I",

"Числовые константы": "N",

"Строковые константы": "S",

"Операции": "O",

"Разделители": "R"

}

def add\_to\_table(table\_name, value):

if value not in tables[table\_name]:

tables[table\_name][value] = len(tables[table\_name]) + 1

return f"{category\_prefixes[table\_name]}{tables[table\_name][value]}"

def lexer(code):

result = []

while code:

if code.startswith(" "):

result.append((add\_to\_table("Разделители", repr("\t")), "\\t"))

code = code[4:]

continue

if code[0] == "\n":

result.append((add\_to\_table("Разделители", repr(code[0])), "\\n"))

code = code[1:]

continue

if code[0] == " ":

code = code[1:]

continue

match = None

if code.startswith("#"):

code = code[code.find("\n") + 1:] if "\n" in code else ""

continue

for pattern, token\_type in TOKEN\_PATTERNS:

match = re.match(pattern, code)

if match:

value = match.group(0)

if value in KEYWORDS\_JAVA:

result.append((add\_to\_table("Ключевые слова", value), value))

else:

result.append((add\_to\_table(token\_type, value), value))

break

if not match:

for op in sorted(OPERATORS\_JAVA, key=len, reverse=True):

if code.startswith(op):

result.append((add\_to\_table("Операции", op), op))

match = op

break

if not match:

for delim in DELIMITERS:

if code.startswith(delim):

result.append((add\_to\_table("Разделители", delim), delim))

match = delim

break

if match:

code = code[len(match):] if isinstance(match, str) else code[len(match.group(0)):]

else:

print(f"Неизвестный символ: {repr(code[0])}")

raise SyntaxError(f"Неизвестный символ: {repr(code[0])}")

return result

def open\_file():

file\_path = filedialog.askopenfilename(filetypes=[("All Files", "\*.\*")])

if file\_path:

try:

with open(file\_path, "r", encoding="utf-8") as file:

code = file.read()

code\_text.delete(1.0, tk.END)

code\_text.insert(tk.END, code)

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка при чтении файла: {e}")

def analyze\_code():

code = code\_text.get(1.0, tk.END)

if code.endswith("\n"):

code = code[:-1]

result = lexer(code)

with open("result.txt", "w", encoding="utf-8") as token\_file:

for token in result:

token\_file.write(f"{token[0]}: {token[1]}\n")

with open("tables\_result.txt", "w", encoding="utf-8") as table\_file:

for table\_name, table in tables.items():

table\_file.write(f"{table\_name}:\n")

for value, index in table.items():

table\_file.write(f"{category\_prefixes[table\_name]}{index} : {value}\n")

table\_file.write("\n")

# Отображаем индексы лексем в каждой строке

result\_text.delete(1.0, tk.END)

lines = code.splitlines(keepends=True)

index\_lines = []

for line in lines:

tokens = lexer(line)

token\_indices = []

for token in tokens:

token\_indices.append(token[0]) # Добавляем индекс лексемы

index\_lines.append(" ".join(token\_indices))

result\_text.insert(tk.END, "\n".join(index\_lines))

root = tk.Tk()

root.title("Лексический анализатор")

notebook = ttk.Notebook(root)

notebook.pack(fill=tk.BOTH, expand=True)

# Экран 1 (Вкладка 1)

tab1 = tk.Frame(notebook)

notebook.add(tab1, text="Лексический анализатор")

left\_frame = tk.Frame(tab1)

left\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

right\_frame = tk.Frame(tab1)

right\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

code\_text = tk.Text(left\_frame, width=60, height=20, wrap=tk.NONE)

code\_text.pack(fill=tk.BOTH, expand=True)

load\_button = tk.Button(left\_frame, text="Загрузить файл", command=open\_file)

load\_button.pack(side=tk.BOTTOM, pady=10, fill=tk.X)

result\_text = tk.Text(right\_frame, width=60, height=20, wrap=tk.NONE)

result\_text.pack(fill=tk.BOTH, expand=True)

analyze\_button = tk.Button(right\_frame, text="Запустить анализатор", command=analyze\_code)

analyze\_button.pack(side=tk.BOTTOM, pady=10, fill=tk.X)

root.mainloop()