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

# Задание

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.

Входное множество:

* Комментарии: //что-то, /\* что-то \*/
* Массивы: int arr[5], int[] arr = new int[5]
* Ввод: Scanner in = new Scanner(System.in); int num = in.nextInt();
* Вывод: System.out.println("Hello world!");
* Примитивные типы данных: int, float, double, char, void, String, boolean
* Арифметические операции: +, -, \*, /, %, ++, --
* Операторы сравнения: ==, !=, >, <, >=, <=
* Логические операторы: &&, ||, !
* Оператор присваивания: =
* Условные операторы: if, else if, else
* Циклы: for, while

# Ход работы

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.

Каждая лексема, обработанная сканером, преобразуется к виду:

<буква><код>,

где: <буква> – это признак класса лексемы (W, I, O, V, 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.

Большинство лексем можно сгруппировать в классы:

1. класс идентификаторов;
2. класс служебных слов;
3. класс констант (числовых или символьных);
4. класс операций (одно-, дву- или многолитерных);
5. класс разделителей (однолитерных или двулитерных).

На вход анализатора подаются символы исходной программы, и каждый символ вызывает изменения состояния сканера. Если символ означает конец разбираемой лексемы, то с переходом связывается семантическая процедура, благодаря которой определяется код лексемы из таблицы или пополняются незаполненный таблицы.

Чтобы оптимизировать разбор программы, в алфавите автомата выделяют несколько подмножеств символов, по которым и работает автомат.

1. буквы {A, …, Z, a, …, z};
2. цифры {0, …, 9};
3. разделители {табуляция, перевод строки, пробел, «(», «)», «,», «.», «:», «;», «[», «]»,«{», «}»};
4. операции {\*, +, -,%, ++, --, /,==, <, <=, !=, =, >, >=,&&, ||, !};
5. точка;
6. одинарная кавычка;
7. двойная кавычка;
8. слэш;
9. конец строки
10. конец файла.

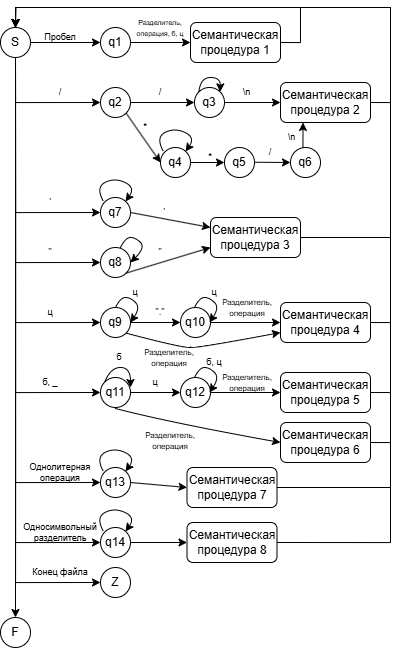


Рисунок 1 – Диаграмма состояний сканера

На диаграмме состояний сканера вершины представляют состояния, а дуги (направленные линии) отображают переходы между этими состояниями. Каждый переход связан с чтением определённого символа из текста входной программы. Таким образом, дуга может быть взвешена (помечена) символом или множеством символов, которые инициируют переход. Если на диаграмме имеется невзвешенная дуга, исходящая из какого-либо состояния, считается, что она активируется любыми символами, кроме тех, которые помечают другие исходящие дуги из этого состояния. Это позволяет избежать излишней перегрузки диаграммы. Если на дуге есть скруглённый прямоугольник, это указывает на семантическую процедуру, выполняемую при переходе. В случае, если переход не связан с семантической процедурой, символ добавляется в буфер, где формируется лексема.

Сканер всегда включает три стандартных состояния:

S – начальное состояние сканера, с которого всегда начинается лексический анализ.

Z – заключительное состояние сканера. Достижение этого состояния свидетельствует о том, что лексический разбор был успешно завершён.

F – состояние ошибки. Если сканер встречает символ, не принадлежащий входному алфавиту, он переходит в состояние ошибки и прекращает разбор.

За семантическими процедурами процессора закреплены следующие функции:

Семантическая процедура 1: При нахождении символа пробела программа переходит к следующей итерации.

Семантическая процедура 2: После считывании символа "/" программа обязательно считает далее "/" или "\*", что означает, что начался комментарий. В любом случае после этих символов могут быть любые символы, которые завершаются "\n". Во втором случае перед "\n" должны идти "\*" и "/".

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

Семантическая процедура 4: Занести сформированное слово в таблицу числовых констант. Сформировать и выдать в выходную последовательность лексему числовой константы.

Семантическая процедура 5: 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.

Семантическая процедура 6: Провести поиск сформированного слова в таблицах служебных слов. Если такое есть служебное слово в таблице служебных слов, то выполнить, сформировать и выдать в выходную последовательность лексему служебного слова.

Семантическая процедура 7: Провести поиск текущего слова в таблице операций. Сформировать и выдать в выходную последовательность лексему операции.

Семантическая процедура 8: Провести поиск сформированного слова в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя.

На рисунке 2 представлена исходная программа на языке Java, на рисунке 3 – последовательность кодов лексем входной программы с игнорированием комментариев и пробелов.

|  |
| --- |
|  |

Рисунок 2 – Код исходной программы на Java

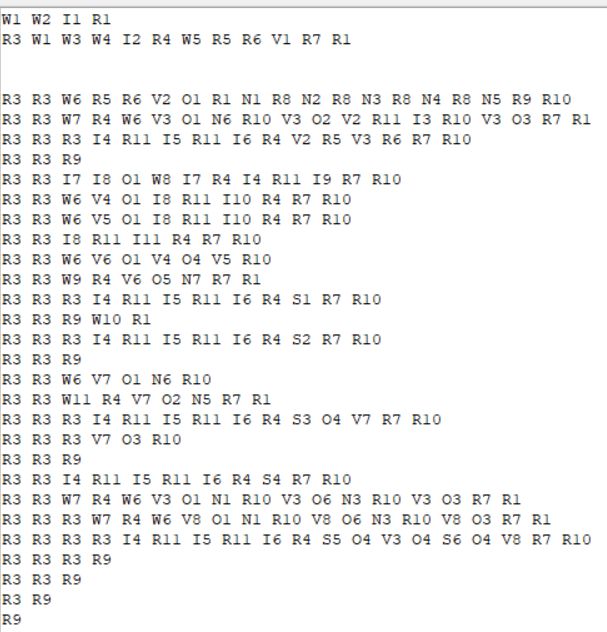
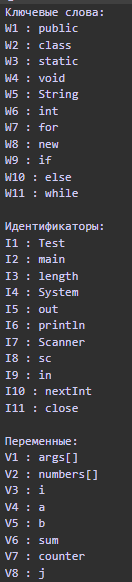


Рисунок 3 – Результат работы программы

В результате работы программы создаются файлы result.txt и tables\_result.txt. На рисунках 4-5 показаны скриншоты файла tables\_result.txt, содержащего лексемы, присутствующие в коде исходной программе на Java.



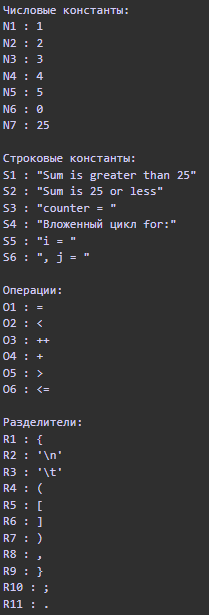


Рисунок 5 – 2 часть таблицы лексем

Рисунок 4 – 1 часть таблицы лексем

На рисунках 6-9 изображено содержимое файла result.txt. Это результат работы программы, сопоставляющий каждой лексеме программы определенную категорию и индекс.

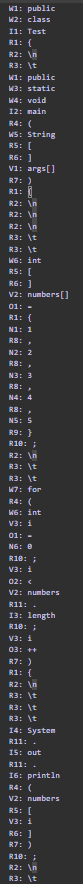


Рисунок 6 – 1 часть файла результатов

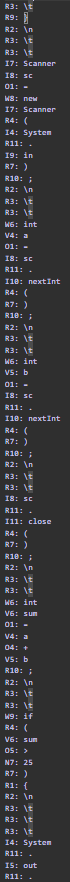


Рисунок 7 – 2 часть файла результатов

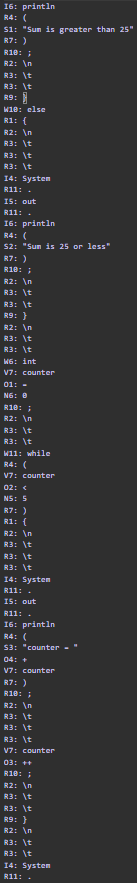


Рисунок 8 – 3 часть файла результатов

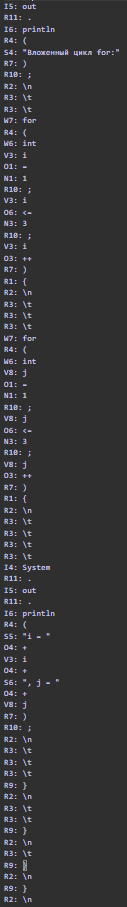


Рисунок 9 – 4 часть файла результатов

# ПРИЛОЖЕНИЕ А

**Листинг программы**

import tkinter as tk

from tkinter import filedialog, messagebox, ttk

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"}

tables = {

"Ключевые слова": {},

"Идентификаторы": {},

"Переменные": {},

"Числовые константы": {},

"Строковые константы": {},

"Операции": {},

"Разделители": {}

}

# Префиксы категорий

category\_prefixes = {

"Ключевые слова": "W",

"Идентификаторы": "I",

"Переменные": "V",

"Числовые константы": "N",

"Строковые константы": "S",

"Операции": "O",

"Разделители": "R"

}

is\_array = {}

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 = []

i = 0

prev\_token = None

is\_declaring\_variable = False

is\_declaring\_array = False

while i < len(code):

char = code[i]

if char == ' ':

i += 1

continue

if char == '\t':

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

i += 1

continue

if char == '\n':

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

i += 1

continue

# Обработка комментариев

if char == '/' and i + 1 < len(code) and code[i + 1] == '/':

# Пропускаем всё до конца строки

while i < len(code) and code[i] != '\n':

i += 1

continue

# Обработка многострочных комментариев (/\* ... \*/)

if char == '/' and i + 1 < len(code) and code[i + 1] == '\*':

i += 2 # Пропускаем /\*

while i + 1 < len(code) and not (code[i] == '\*' and code[i + 1] == '/'):

i += 1

if i + 1 < len(code):

i += 2

continue

# Строковые константы (двойные кавычки)

if char == '"':

start = i

i += 1

while i < len(code) and code[i] != '"':

i += 1

value = code[start:i+1]

result.append((add\_to\_table("Строковые константы", value), value))

i += 1

continue

# Строковые константы (одинарные кавычки)

if char == "'":

start = i

i += 1

while i < len(code) and code[i] != "'":

i += 1

value = code[start:i+1]

result.append((add\_to\_table("Строковые константы", value), value))

i += 1

continue

# Числовые константы

if char.isdigit():

start = i

while i < len(code) and (code[i].isdigit() or code[i] == '.'):

i += 1

value = code[start:i]

result.append((add\_to\_table("Числовые константы", value), value))

continue

# Ключевые слова, идентификаторы и переменные

if char.isalpha() or char == '\_':

start = i

while i < len(code) and (code[i].isalnum() or code[i] == '\_'):

i += 1

value = code[start:i]

if value in KEYWORDS\_JAVA:

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

if value in {"int", "String", "float", "double", "boolean", "char"}:

if i < len(code) and code[i] == '[':

is\_declaring\_array = True

else:

is\_declaring\_variable = True

else:

if is\_declaring\_variable:

if value not in tables["Переменные"]:

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

is\_declaring\_variable = False

elif is\_declaring\_array:

if value not in tables["Переменные"]:

result.append((add\_to\_table("Переменные", value), f"{value}[]"))

is\_array[value] = True

is\_declaring\_array = False

# Проверяем, есть ли уже такая переменная в таблице

if value in tables["Переменные"]:

result.append((f"V{tables['Переменные'][value]}", value))

else:

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

continue

# Операции

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

if code[i:].startswith(op):

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

i += len(op)

break

else:

# Разделители (оставшиеся символы)

for delim in sorted(DELIMITERS, key=len, reverse=True):

if code[i:].startswith(delim):

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

i += len(delim)

break

else:

print(f"Неизвестный символ: {repr(char)}")

raise SyntaxError(f"Неизвестный символ: {repr(char)}")

filtered\_result = []

prev\_token = None

for token in result:

if prev\_token and prev\_token[0].startswith("V") and prev\_token[0] == token[0]:

continue # Пропускаем этот токен, если он совпадает с предыдущим

filtered\_result.append(token)

prev\_token = token

return filtered\_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():

if table\_name == "Переменные" and value in is\_array:

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

else:

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

table\_file.write("\n")

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

for token in result:

if token[1] == '\\n':

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

else:

result\_text.insert(tk.END, f"{token[0]} ")

root = tk.Tk()

root.title("Разработка транслятора")

notebook = ttk.Notebook(root)

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

tab1 = tk.Frame(notebook)

notebook.add(tab1, text="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=70, 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=70, 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()