Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы №1

по дисциплине “Методы разработки трансляторов”

Выполнил: ст. гр. 36/2

Баева Д. Н.

Проверил: Вишняков Ю.М

Краснодар

2023

**Содержание**

[1 Вариант задания 4](#_Toc97893076)

[2 Основные понятия лексического анализа 6](#_Toc97893077)

[3 Функции и таблицы лексического анализа 8](#_Toc97893078)

[4 Диаграмма состояний лексического процессора 10](#_Toc97893079)

[5 Результаты экспериментов 17](#_Toc97893080)

[Приложение А Описание правил записи элементов заданного входного языка 21](#_Toc97893081)

[Приложение Б Листинг программы и комментарии к нему 25](#_Toc97893082)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 27 | R | С++ |

Разработать лексический анализатор, корректно распознающий все лексемы и формирующего таблицы лексем и внутреннее представление проанализированного текста.

На вход программы подается файл, содержащий текст на входном языке программирования. Результатом работы программы должен быть файл, содержащий последовательность кодов лексем входной программы, а также один или несколько файлов, содержащих все таблицы лексем.

Отчет по работе содержит описание правил записи элементов заданного входного языка, стандартные таблицы лексем, диаграмма состояний сканера, листинг программы и комментарии к нему, пример лексического разбора.

**2 Основные понятия лексического анализа**

Лексический анализ всегда предшествует синтаксическому анализу и заключается в предварительной обработке программы и ее перекодировании к виду, удобному для синтаксического анализа. Лексический анализ принято отделять от синтаксического анализа для сокращения времени компиляции, поскольку синтаксис (морфология) слов всегда проще, чем синтаксис программ.

Лексический анализ сводится к разбиению текста программы на лексические единицы (слова, лексемы) – конструкции, которые выступают в качестве терминальных символов для синтаксического анализа. Лексемы еще иногда называют символами или атомами.

Большинство лексем в языках программирования можно сгруппировать в следующие 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).

В лексическом анализе лексема представляется в виде пары (класс, значение).

**3 Функции и таблицы лексического анализа**

Основная функция лексического анализатора (сканера) состоит в выделении лексем из исходной программы и передаче их синтаксическому анализатору в некотором внутреннем представлении. При этом лексический анализатор использует таблицы, соответствующие классам лексем. Таблицы служебных слов, разделителей и операций определяются входным языком и должны быть сформированы при построении сканера, они являются постоянными и неизменяемыми. Таблицы идентификаторов и констант являются временными и создаются непосредственно в процессе лексического разбора исходной программы.

Комментарии в процессе лексического анализа игнорируются, так как они специфицируют исходную программу только для программиста и не влияют на семантику программы.

Для рассматриваемого языка сформированные таблицы служебных слов (таблица 2), таблица операций (таблица 3) и таблица разделителей (таблица 4) имеют вид:

Таблица 2 – Таблица служебных слов

|  |  |
| --- | --- |
| Служебное слово | Код |
| if | 1 |
| else | 2 |
| print | 3 |
| for | 4 |
| in | 5 |
| while | 6 |
| scan | 7 |
| function | 8 |
| c | 9 |

Таблица 3 – Таблица операций

|  |  |
| --- | --- |
| Операция | Код |
| ! | 1 |
| != | 2 |
| %% (остаток от деления) | 3 |
| \* | 4 |
| ^ (возведение в степень) | 5 |
| + | 6 |
| - | 7 |
| <- | 8 |
| / | 9 |
| < | 10 |
| <= | 11 |
| = | 12 |
| == | 13 |
| > | 14 |
| >= | 15 |
| & | 16 |
| | | 17 |

Таблица 4 – Таблица разделителей

|  |  |
| --- | --- |
| Разделитель | Код |
| табуляция | 1 |
| перевод строки | 2 |
| пробел | 3 |
| ( | 4 |
| ) | 5 |
| , | 6 |
| . | 7 |
| : | 8 |
| ; | 9 |
| [ | 10 |
| ] | 11 |
| { | 12 |
| } | 13 |

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 лексическом разборе входной цепочки пробелы не кодируются как разделители и в выходную цепочку не попадают. Это связано с тем, что в исходном тексте программы пробелы служат только для структурирования текста программы и не несут дополнительной смысловой нагрузки.

**4 Диаграмма состояний лексического процессора**

Лексический анализатор представляет собой процессор или, иначе говоря, сканер (конечный автомат) для разбора и классификации лексем, связанный с некоторыми семантическими процедурами. На вход такого процессора последовательно подаются символы исходной программы, причем каждый входной символ вызывает изменение состояния сканера. Если анализируемый символ означает конец разбираемой лексемы, то с переходом связывается некоторая семантическая процедура, позволяющая либо определить код лексемы по таблице (в случае служебных слов, разделителей, операций), либо пополнить таблицы (для констант и идентификаторов), а затем в выходную строку выдать очередной код лексемы во внутреннем представлении.

Для эффективного разбора и оптимизации сканера, а также для его программной реализации в алфавите автомата выделяют несколько подмножеств символов, по которым сканер выполняет одинаковые переходы и действия. В нашем случае такими подмножествами являются:

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

С учетом вышесказанного диаграмма состояний сканера для рассматриваемого языка имеет вид, представленный на рисунках 1-7.

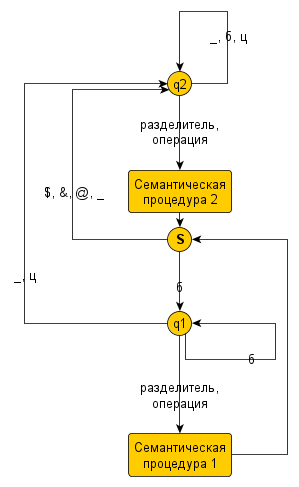


Рисунок 1 – Диаграмма состояний сканера для распознавания служебного слова или идентификатора

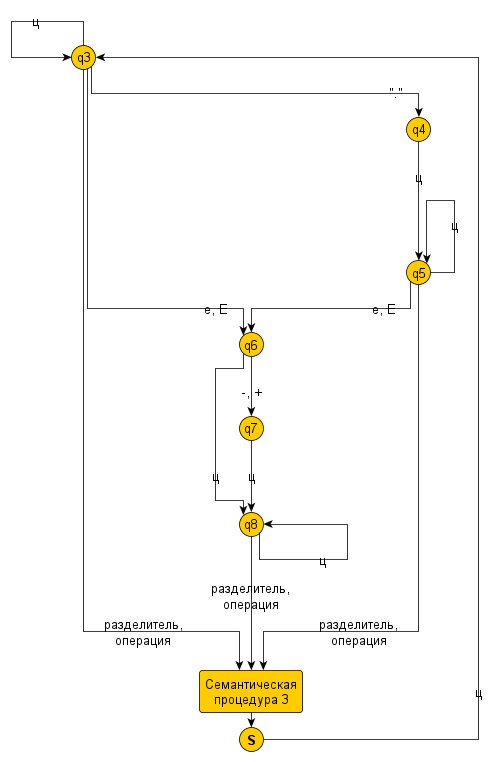


Рисунок 2 – Диаграмма состояний сканера для распознавания числовой константы

Изображение выглядит как текст, знак

Автоматически созданное описание

Рисунок 3 – Диаграмма состояний сканера для распознавания символьной константы

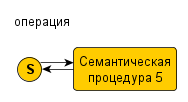


Рисунок 4 – Диаграмма состояний сканера для распознавания операции

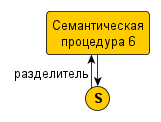


Рисунок 5 – Диаграмма состояний сканера для распознавания разделителя

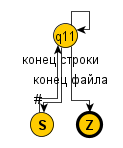


Рисунок 6 – Диаграмма состояний сканера для распознавания комментария

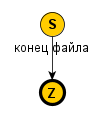


Рисунок 7 – Диаграмма состояний сканера для определения конца файла

На данной диаграмме состояния сканера представлены вершинами, а переходы между состояниями – дугами (направленными линиями). Каждый переход связан с чтением очередного символа их 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. Скругленный 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:

1. S – начальное состояние сканера. Лексический разбор всегда начинается из этого состояния.
2. Z – заключительное состояние сканера. Если в процессе разбора достигнуто данное состояние, это означает, что разбор успешно завершен.
3. 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 таблицах служебных слов, операций. Если такое служебное слово в таблице служебных слов, операция в таблице операций не найдены, то выполнить Семантическую процедуру 2, иначе сформировать и выдать в выходную последовательность лексему служебного слова или операции во внутреннем представлении.

Семантическая процедура 2: 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.

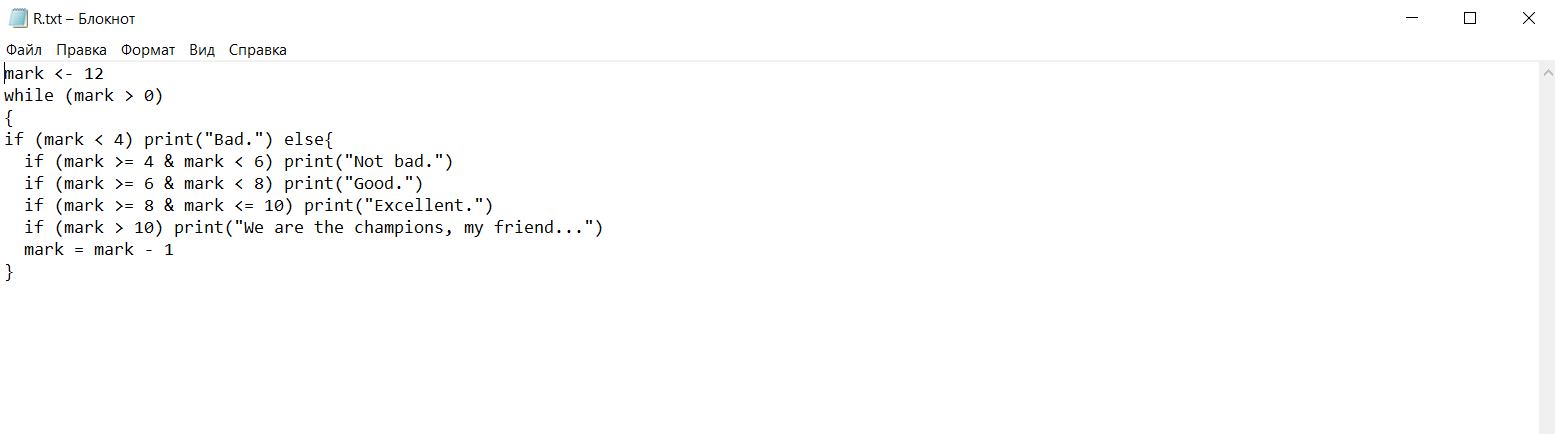
Семантическая процедура 3: Занести сформированное слово в таблицу числовых констант. Сформировать и выдать в выходную последовательность лексему числовой константы во внутреннем представлении.

Семантическая процедура 4: Занести сформированное слово в таблицу символьных констант. Сформировать и выдать в выходную последовательность лексему символьной константы во внутреннем представлении.

Семантическая процедура 5: Провести поиск сформированного слова в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 6: Провести поиск текущего символа в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении

**5 Результаты экспериментов**

Пример работы программы для программы для тестирования: на рисунке 8 представлен скриншот файла, содержащего текст на входном языке программирования, на рисунке 9 – скриншот файла, содержащего последовательность кодов лексем входной программы, на рисунках   
10-15 – скриншоты файлов, содержащие все лексемы.

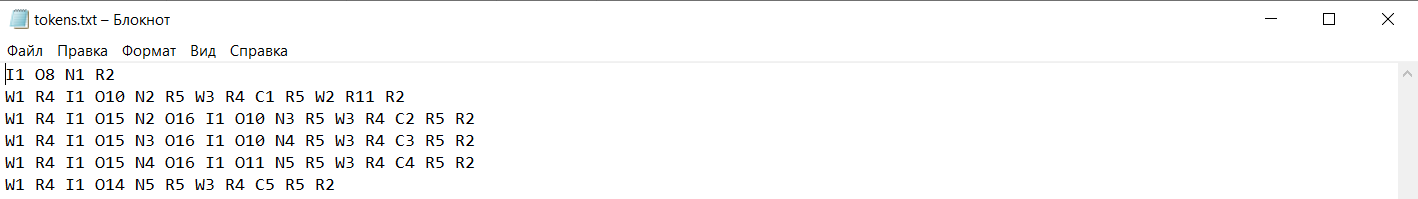
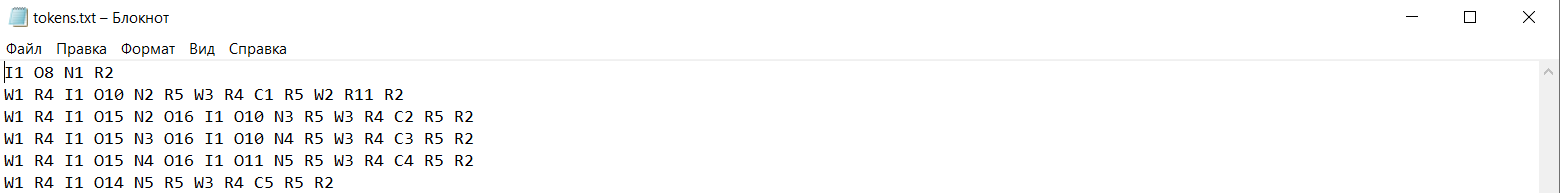
Рисунок 8 – Скриншот файла, содержащего текст на входном языке программирования

Рисунок 9 – Скриншот файла, содержащего последовательность кодов лексем входной программы

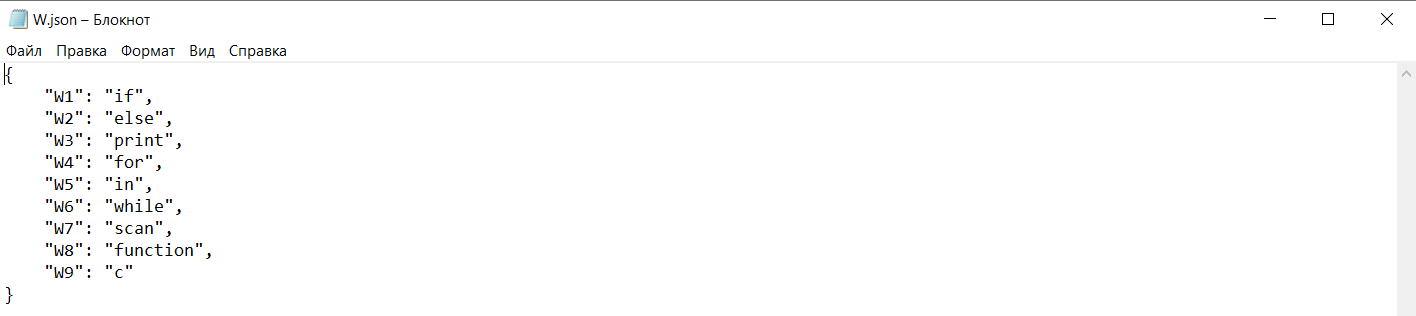


Рисунок 10 – Скриншот файла, содержащего лексемы служебных слов

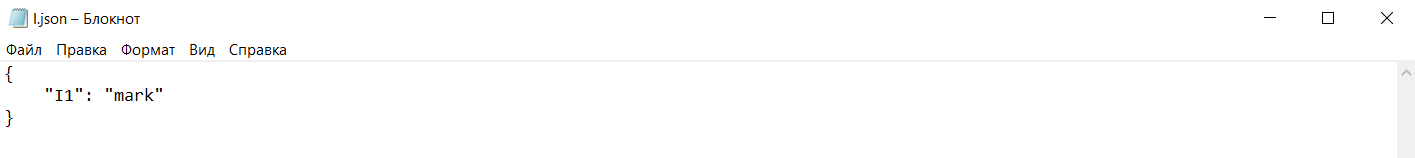
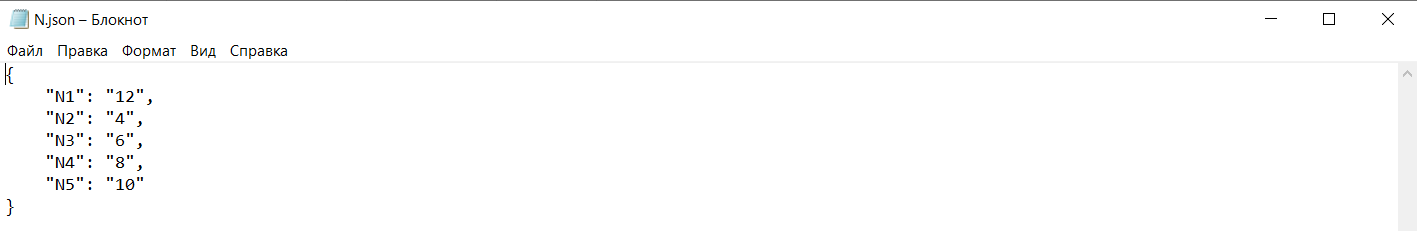
Рисунок 11 – Скриншот файла, содержащего лексемы идентификаторов

Рисунок 12 – Скриншот файла, содержащего лексемы числовых констант

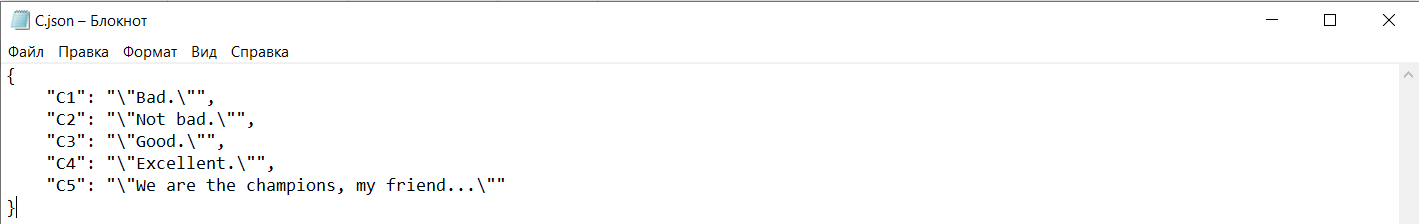


Рисунок 13 – Скриншот файла, содержащего лексемы символьных констант

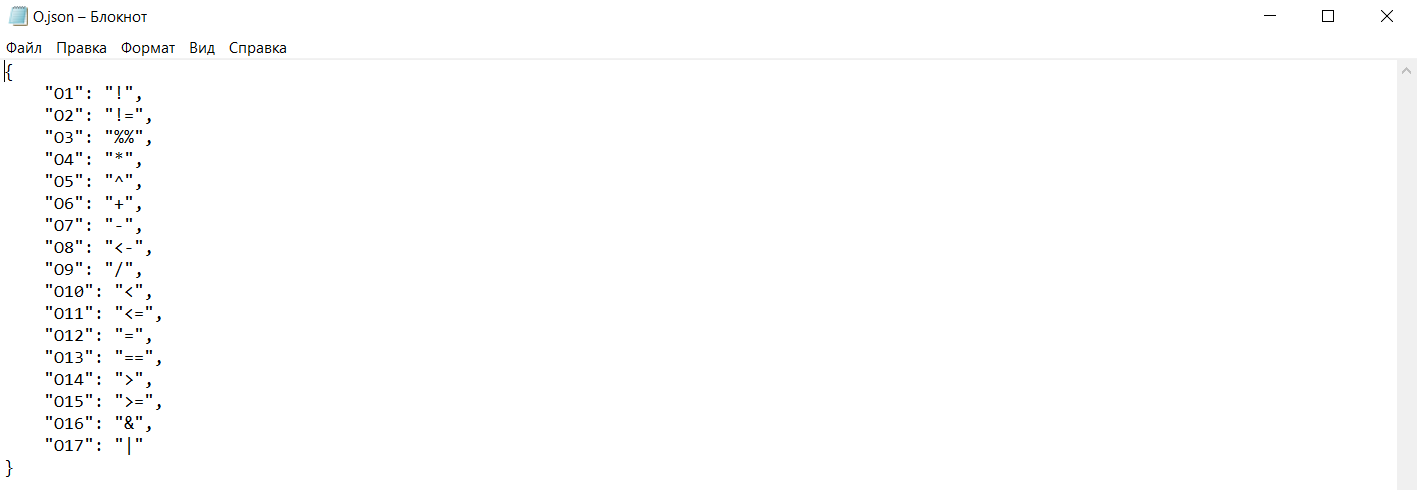


Рисунок 14 – Скриншот файла, содержащего лексемы операций

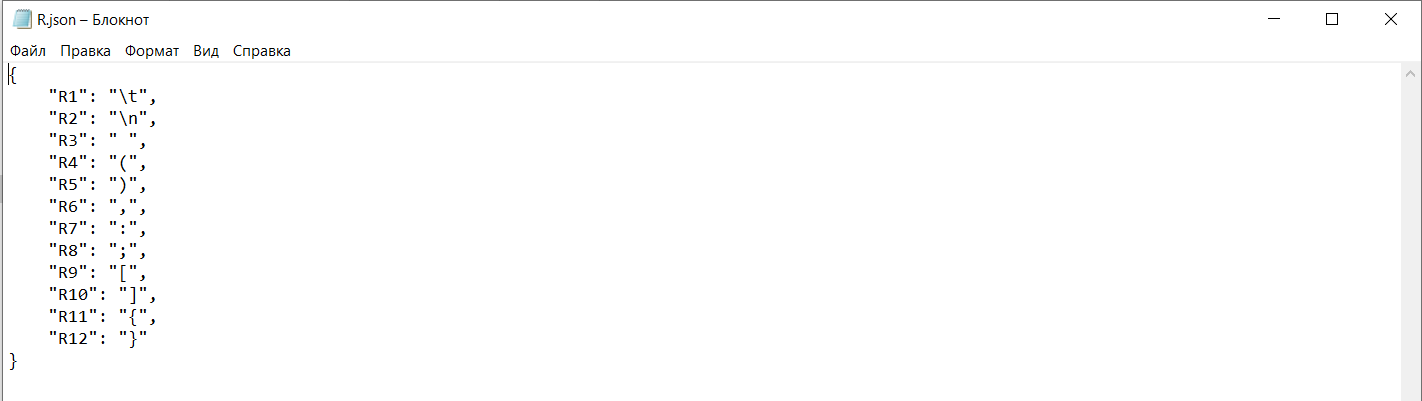


Рисунок 15 – Скриншот файла, содержащего лексемы разделителей

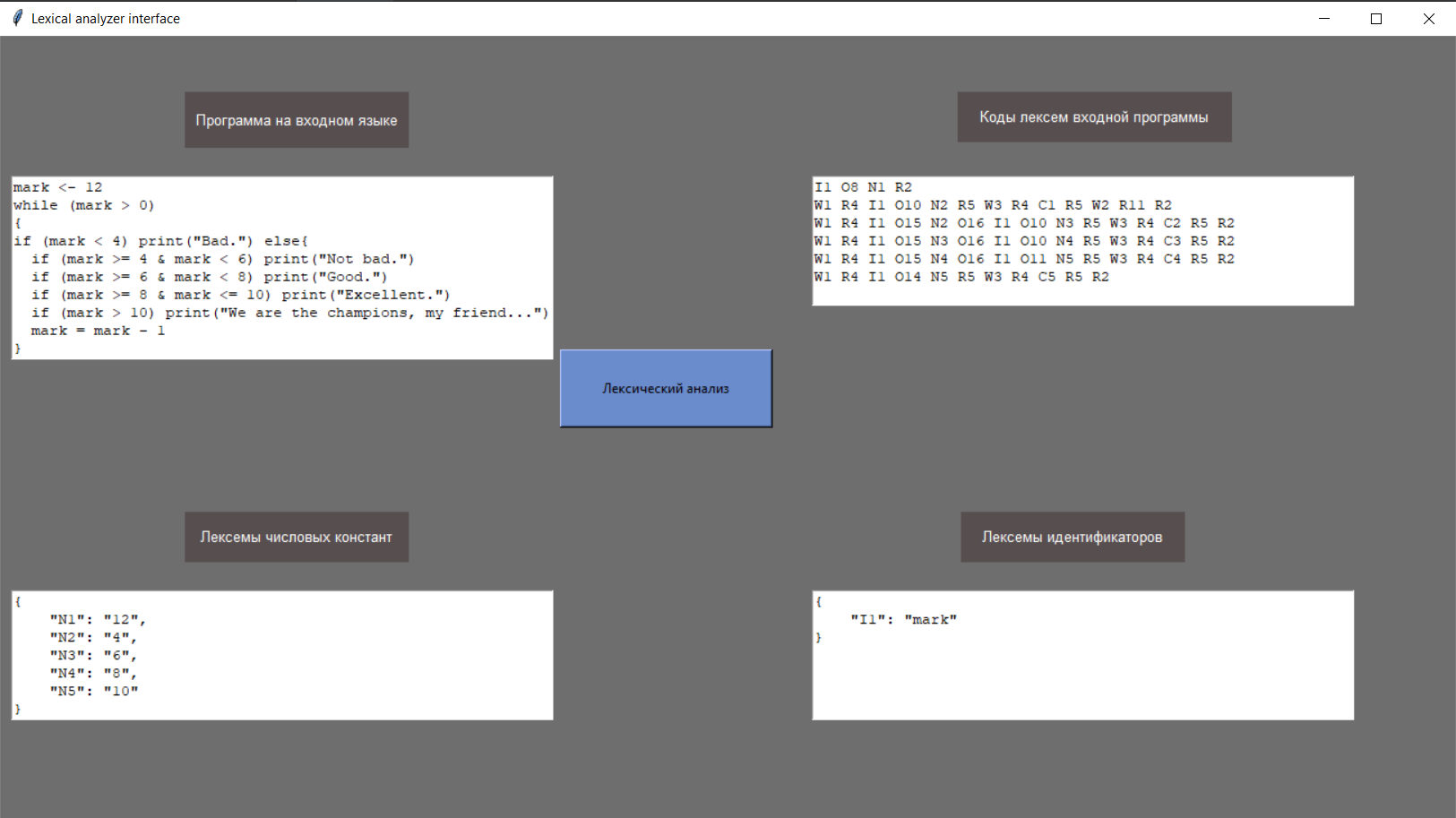


Рисунок 16 – Скриншот интерфейса после выполнения программы

**Приложение А   
Описание правил записи элементов заданного входного языка**

***Язык программирования R (версия 3.3.2)***

**Идентификаторы**

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчёркивания и точки.

**Числовые константы целого типа**

Произвольная последовательность цифр без знака.

**Числовые константы вещественного типа, представленные с фиксированной точкой**

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

**Числовые константы вещественного типа, представленные с плавающей точкой**

Последовательность, включающая цифры, десятичную точку (необязательную), символ «е», а также знак «+» или «-» (необязательный) вида

1.23е-25

1.23е+25

1.23е2

1е-78

1е67

**Символьные (строковые) константы**

Набор символов, возможно пустой, заключённый в одиночные или двойные кавычки.

‘Это строковая константа’

“Это тоже строковая константа”

**Переменные с индексами (векторы и элементы векторов)**

Идентификатор, после которого в квадратных скобках стоит выражение-индекс, вида:

Abc[12]

C[1+i]

**Комментарии (строчные и блочные)**

Только строчные – последовательность символов от знака «#» до конца строки:

X=45 #Это комментарий

**Обращения к функциям пользователя**

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделённых запятыми. Отсутствие аргументов не допускается:

F(12, 4, i)

f(av-6)

**Арифметические операции**

Сложение +

Вычитание -

Умножение \*

Деление /

Возведение в степень ^

**Операции сравнения**

Меньше <

Больше >

Равно ==

Не равно !=

Меньше или равно <=

Больше или равно >=

**Оператор присваивания**

Имеет вид «=» или «<-». Слева стоит идентификатор или элемент вектора, а справа – объект. Пример:

A<-b+с;

b[2]=12;

**Операторы блока**

{ – начало блока

операторы

} - конец блока

**Структура программы**

Программы на R состоят из выражений (операторов), выражения состоят из объектов и функций. Объекты (векторы, списки, функции). Имена переменных называются символами. Среда – это набор символов. Функции – это объекты, которые принимают на вход объекты (аргументы) и выдают объекты.

**Операторы описания данных (идентификаторов и массивов)**

Операторы описания данных в языке отсутствуют.

**Операторы описания функций**

Функции имеют заголовок вида

function <идентификатор> (<список формальных параметров>)

и тело – список операторов, заключённый в операторы блока

{ … }

Пример:

my\_mean <- function(x)

{

avg <- sum(x) / length(x)

return(avg)

}

**Оператор условного перехода**

Начинается с ключевого слова «if», имеет полный и неполный формат:

if (*логическое выражение*)

{

*операторы*

}

if (*логическое выражение*)

{

*Операторы\_1*

}

else

{

*Операторы\_2*

}

**Цикл с предусловием**

Цикл с предусловием имеет следующий вид: while (<условие>) {<операторы>}.

Пример:

While (i <= 3)

{

print(c(i,p[i]))

i<-i+1

}

**Приложение Б   
Листинг программы и комментарии к нему**

import json  
import tkinter  
from tkinter import \*  
  
  
SERVICE\_WORDS = ['if', 'else','print','for',\  
 'in', 'while', 'scan', 'function','c']  
OPERATIONS = ['!', '!=', '%%', '\*', '^', '+', '-', '<-', '/', '<', '<=', '=',\  
 '==', '>', '>=','&','|']  
SEPARATORS = ['\t','\n', ' ', '(', ')', ',', ':', ';', '[', ']', '{', '}']  
def act():  
 def check(tokens, token\_class, token\_value):  
 if not (token\_value in tokens[token\_class]):  
 token\_code = str(len(tokens[token\_class]) + 1)  
 tokens[token\_class][token\_value] = token\_class + token\_code  
  
 def get\_operation(input\_sequence, i):  
 for k in range(2, 0, -1):  
 if i + k < len(input\_sequence):  
 buffer = input\_sequence[i:i + k]  
 if buffer in OPERATIONS:  
 return buffer  
 return ''  
  
 def get\_separator(input\_sequence, i):  
 buffer = input\_sequence[i]  
 if buffer in SEPARATORS:  
 return buffer  
 return ''  
  
 *# лексемы* tokens = {'W': {}, 'I': {}, 'O': {}, 'R': {}, 'N': {}, 'C': {}}  
  
 *# заполнение словаря со служебными словами* for service\_word in SERVICE\_WORDS:  
 check(tokens, 'W', service\_word)  
 *# заполнение словаря с операциями* for operation in OPERATIONS:  
 check(tokens, 'O', operation)  
 *# заполнение словаря с разделителями* for separator in SEPARATORS:  
 check(tokens, 'R', separator)  
  
 *# файл, содержащий текст на входном языке программирования* f = open('R.txt', 'r')  
 input\_sequence = f.read()  
 f.close()  
  
 i = 0  
 state = 'S'  
 output\_sequence = buffer = ''  
 while i < len(input\_sequence):  
 symbol = input\_sequence[i]  
 operation = get\_operation(input\_sequence, i)  
 separator = get\_separator(input\_sequence, i)  
 if state == 'S':  
 buffer = ''  
 if symbol.isalpha():  
 state = 'q1'  
 buffer += symbol  
 elif symbol.isdigit():  
 state = 'q3'  
 buffer += symbol  
 elif symbol == "'":  
 state = 'q9'  
 buffer += symbol  
 elif symbol == '"':  
 state = 'q10'  
 buffer += symbol  
 elif symbol == '/':  
 state = 'q11'  
 elif operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 elif separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 elif i == len(input\_sequence) - 1:  
 state = 'Z'  
 elif state == 'q1':  
 if symbol.isalpha():  
 buffer += symbol  
 elif symbol.isdigit():  
 state = 'q2'  
 buffer += symbol  
 else:  
 if operation or separator:  
 if buffer in SERVICE\_WORDS:  
 output\_sequence += tokens['W'][buffer] + ' '  
 elif buffer in OPERATIONS:  
 output\_sequence += tokens['O'][buffer] + ' '  
 else:  
 check(tokens, 'I', buffer)  
 output\_sequence += tokens['I'][buffer] + ' '  
 if operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 if separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 state = 'S'  
 elif state == 'q2':  
 if symbol.isalnum():  
 buffer += symbol  
 else:  
 if operation or separator:  
 check(tokens, 'I', buffer)  
 output\_sequence += tokens['I'][buffer] + ' '  
 if operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 if separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 state = 'S'  
 elif state == 'q3':  
 if symbol.isdigit():  
 buffer += symbol  
 elif symbol == '.':  
 state = 'q4'  
 buffer += symbol  
 elif symbol == 'e' or symbol == 'E':  
 state = 'q6'  
 buffer += symbol  
 else:  
 if operation or separator:  
 check(tokens, 'N', buffer)  
 output\_sequence += tokens['N'][buffer] + ' '  
 if operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 if separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 state = 'S'  
 elif state == 'q4':  
 if symbol.isdigit():  
 state = 'q5'  
 buffer += symbol  
 elif state == 'q5':  
 if symbol.isdigit():  
 buffer += symbol  
 elif symbol == 'e' or symbol == 'E':  
 state = 'q6'  
 buffer += symbol  
 else:  
 if operation or separator:  
 check(tokens, 'N', buffer)  
 output\_sequence += tokens['N'][buffer] + ' '  
 if operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 if separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 state = 'S'  
 elif state == 'q6':  
 if symbol == '-' or symbol == '+':  
 state = 'q7'  
 buffer += symbol  
 elif symbol.isdigit():  
 state = 'q8'  
 buffer += symbol  
 elif state == 'q7':  
 if symbol.isdigit():  
 state = 'q8'  
 buffer += symbol  
 elif state == 'q8':  
 if symbol.isdigit():  
 buffer += symbol  
 else:  
 if operation or separator:  
 check(tokens, 'N', buffer)  
 output\_sequence += tokens['N'][buffer] + ' '  
 if operation:  
 check(tokens, 'O', operation)  
 output\_sequence += tokens['O'][operation] + ' '  
 i += len(operation) - 1  
 if separator:  
 if separator != ' ':  
 check(tokens, 'R', separator)  
 output\_sequence += tokens['R'][separator]  
 if separator == '\n':  
 output\_sequence += '\n'  
 else:  
 output\_sequence += ' '  
 state = 'S'  
 elif state == 'q9':  
 if symbol != "'":  
 buffer += symbol  
 elif symbol == "'":  
 buffer += symbol  
 check(tokens, 'C', buffer)  
 output\_sequence += tokens['C'][buffer] + ' '  
 state = 'S'  
 elif state == 'q10':  
 if symbol != '"':  
 buffer += symbol  
 elif symbol == '"':  
 buffer += symbol  
 check(tokens, 'C', buffer)  
 output\_sequence += tokens['C'][buffer] + ' '  
 state = 'S'  
 elif state == 'q11':  
 if symbol == '/':  
 state = 'q12'  
 elif symbol == '\*':  
 state = 'q13'  
 elif state == 'q12':  
 if symbol == '\n':  
 state = 'S'  
 elif i == len(input\_sequence) - 1:  
 state = 'Z'  
 elif state == 'q13':  
 if symbol == '\*':  
 state = 'q14'  
 elif state == 'q14':  
 if symbol == '/':  
 state = 'q15'  
 elif state == 'q15':  
 if symbol == '\n':  
 state = 'S'  
 elif i == len(input\_sequence) - 1:  
 state = 'Z'  
 i += 1  
  
 *# файлы, содержащие все таблицы лексем* for token\_class in tokens.keys():  
 with open('%s.json' % token\_class, 'w') as write\_file:  
 data = {val: key for key, val in tokens[token\_class].items()}  
 json.dump(data, write\_file, indent=4, ensure\_ascii=False)  
  
 *# файл, содержащий последовательность кодов лексем входной программы* f = open('tokens.txt', 'w')  
 f.write(output\_sequence)  
 f.close()  
  
def lex\_anal():  
 label3 = tkinter.Label(window, text='Коды лексем входной программы', font=("Arial", 10), foreground="white",background="#574f4f")  
 label3.place(x=855, y=50, width=245, height=45)  
 token\_text = open('tokens.txt',encoding='utf-8').readlines()  
 token\_text = ''.join(token\_text)  
 textline = Text(window, height=7, width=60)  
 textline.insert(1.0, token\_text)  
 textline.place(x=725, y=125)  
 i\_text = open('I.json',encoding='utf-8').readlines()  
 i\_text = ''.join(i\_text)  
 textline = Text(window, height=7, width=60)  
 textline.insert(1.0, i\_text)  
 textline.place(x=725, y=495)  
 label4 = tkinter.Label(window, text='Лексемы идентификаторов', font=("Arial", 10), foreground="white",background="#574f4f")  
 label4.place(x=858, y=425, width=200, height=45)  
 n\_text = open('N.json',encoding='utf-8').readlines()  
 n\_text = ''.join(n\_text)  
 textline = Text(window, height=7, width=60)  
 textline.insert(1.0, n\_text)  
 textline.place(x=10, y=495)  
 label5 = tkinter.Label(window, text='Лексемы числовых констант', font=("Arial", 10), foreground="white",background="#574f4f")  
 label5.place(x=165, y=425, width=200, height=45)  
  
  
window = tkinter.Tk()  
window.geometry('1300x700')  
window.title("Lexical analyzer interface")  
window.configure(bg='#6e6e6e')  
  
  
label2 = tkinter.Label(window, text='Программа на входном языке',font=("Arial", 10),foreground="white", background="#574f4f")  
label2.place(x=165, y=50, width=200, height=50)  
  
  
text = open('R.txt',encoding='utf-8').readlines()  
text=''.join(text)  
textline=Text(window,height=10, width=60)  
textline.insert(1.0,text)  
textline.place(x=10, y=125)  
  
button = tkinter.Button(window, text='Лексический анализ', bg='#6a8bcc',command=lex\_anal)  
button.place(x=500, y=280, width=190, height=70)  
  
window.mainloop()