МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра вычислительной техники



**ОТЧЕТ**

**По лабораторной работе №2**

**Лексика языков программирования. Конечные автоматы без памяти для обнаружения слов в тексте программы**

**по дисциплине: «*Теория формальных языков и компиляторов*»**

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студент гр. АВТ-709, АВТФ | *к.т.н., доцент* |
| *Убушеев Т. О.* | *Малявко А.А.* |
| «\_\_\_» \_\_\_\_\_\_ 2020 г. | «\_\_\_» \_\_\_\_\_\_ 2020 г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

г. Новосибирск, 2020

# Цель работы

Изучение конечных автоматов (КА) без памяти, способов определения КА – канонического, графового и табличного, методов построения недетерминированного КА по системе регулярных выражений, методов эквивалентных преобразований недетерминированных КА в оптимальные полностью определенные КА – лексические акцепторы.

# Постановка задачи

1. Используя пакет ВебТрансЛаб, освоить:

- создание лексических правил на языке регулярных выражений (РВ);

- использование операций «+, \*, ?, конкатенации и выбора» языка РВ для построения сложных регулярных выражений;

- преобразование простой системы РВ в одноавтоматный лексический акцептор;

- добавление правил и действий в систему РВ для построения мультиавтоматного лексического акцептора;

2. Разработать (доработать разработанный при выполнении работы №1) фрагмент системы регулярных выражений для всех (или выбранной самостоятельно части) групп слов языка, определенного заданием на курсовую работу. Построить по этому фрагменту:

- программный модуль, управляемый графом состояний и переходов;

- программный модуль, управляемый таблично;

3. Изучить структуру программных модулей, построенных ВебТрансЛабом, изучить алгоритмы работы лексического акцептора для графового и табличного способов реализации КА, сравнить реализации конечных автоматов, управляемых различными способами, между собой, оценить их затраты времени в шагах на распознавание слов;

4. Изучить по тексту программного модуля способ реализации вызова действий, определенных в лексических правилах и алгоритм работы формирователя лексем.

5. Проверить функционирование конечных автоматов, построенных ВебТрансЛабом (подготовить тестовый пример программы на языке, заданном на курсовую работу, запустить каждый автомат на выполнение, протрассировать вручную работу лексического акцептора в графовой и табличной реализации, убедиться в работоспособности автоматов, в противном случае – доработать систему РВ и добиться правильного функционирования лексического акцептора).

Вариант: 23112143

**Описание варианта:**

1. Идентификатор:

<бБ><пЦ><бБ>

Примеры: *d23U, N0Nq, x15y.*

1. Константы:

целые по основаниям 4,8 и 10 (*4x1, 8x5, 456*);

вещественные (*123.456*);

строковые (*'abc', "ABC").*

1. Объявления примитивных типов (целый, вещественный, символьный):

long[\_u]

number

char

Примеры:

*long l07g* – целый;

*number n0n23r* – вещественный;

*char c4r* – символьный.

1. Оператор присваивания:

<И> = <В>;

Примеры:

*n0n = 123.321;*

*l07g = 12 + 34;*

1. Условный оператор

if (<ЛВ>) <ОБ> [ ifnot <ОБ> ] fi

Пример:

# *if (l0w < h1g)*

# *{*

# *long i9t = 0;*

*i9t = 23;*

# *}*

# *fi*

# *if (i9t <= 100) i9t = 100;*

*ifnot i9t = 0; fi*

1. Оператор цикла:

loop <ОБ> until <ЛВ>

Пример:

*loop d123s = d123s + 100; until d123s <= 1000;*

1. Оператор переключателя:

switch <В> { case <К> : <ОБ> [gout;] …[ case () <ОБ> ] }

Пример:

*switch i9t { case 1 : i9t = 0; gout; case 2 : i9t = 3; }*

*switch i9t { case 1 : i9t =0; gout; case 2 : i9t = 3; gout; case 3 : {*

*t3p = i9t;*

*i9t = i92t;*

*i92t = t3p;*

*}}*

# Ход работы

**Основные понятия**

Произвольное выражение – это совокупность идентификаторов, знаков математических операций и круглых скобок подчиняемая математическим законам.

Логическое выражение – это выражение результат, которого можно интерпретировать как истину или ложь.

Константа – это число, символ или строка символов, которая в программе используется, как значение. Значение константы нельзя изменить.

Оператор – один выполняемый оператор (присваивания, условный оператор, оператор цикла, переключатель)

Блок – это набор идущих подряд операторов, обрамлённый фигурными скобками.

Функция – это фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы.

Возвращаемое значение – это значение, которое вернет функция после выполнения. Тип возвращаемого выражения должен совпадать с типом функции.

**Функции**

Для определения функции используется следующая конструкция:  
*<тип> <идентификатор> ([<аргументы>]) {[<Оператор\_или\_блок>]}*

Тело функции находится в фигурных скобочках. Для передачи агрументов в функцию их нужно вписать в поле <аргументы> через запятую.

Для возвращения значения из функции используется ключевое слово *return*. При этом нужно указать тип возвращаемого значения перед идентификатором функции при ее определении. Если функция ничего не возвращает, то стоит указать тип *void*.

Пример:

*number p123r(number n104s)*

*{*

*return n104s + 143;*

*}*

Пример вызова вышеописанной функции:

*number f1t = p123r(123.321);*

**Знаки пунктуации и операторы**

Знаки пунктуации и специальные символы из набора символов имеют самое разное предназначение, от организации текста программы до определения задач, которые будут выполнены компилятором или скомпилированной программой.

Существуют следующие знаки пунктуации и операторы:

; точка с запятой

: двоеточие

, запятая

“ двойные кавычки

‘ одинарные кавычки

( левая круглая скобка

) правая круглая скобка

[ левая прямоугольная скобка

] правая прямоугольная скобка

{ левая фигурная скобка

} правая фигурная скобка

< меньше

> больше

<= меньше равно

>= больше равно  
!= не равно

/ знак деления

% знак остатка от деления

+ плюс

\* знак умножения

- минус

== равно

++ инкремент

- - декремент

**Комментарии**

Комментарии — это последовательность символов, игнорируемая компилятором.

Комментарии могут быть везде, где допустимо появление разделительных символов.

Пример:

*log l0007g; /\*Комментарий\*/*

**Ключевые слова**

Ключевые слова - это заранее определенные идентификаторы, которые имеют для компилятора специальное значение. Их можно использовать только так, как они определены. Имя элемента программы не может совпадать по написанию с ключевым словом. Переопределить ключевые слова нельзя.

Существуют следующие ключевые слова:  
*long*

*long \_u*

*number*

*char*

*void*

*return*

*loop*

*until*

*switch*

*case*

*gout*

*if*

*ifnot*

*fi*

**Тестоваяпрограмма*:***

*long m41n()*

*{*

*long n0n = 10;*

*long m44s[n0n] = { 1, 25, 6, 32, 43, 5, 96, 23, 4, 55 };*

*long m1n = 0; /\* для записи минимального значения \*/*

*long b8f = 0; /\* для обмена значениями \*/*

*/\* Начало сортировки \*/*

*long i1i = 0;*

*loop {*

*m1n = i1i; /\* запомним номер текущей ячейки, как ячейки с минимальным значением\*/*

*/\* в цикле найдем реальный номер ячейки с минимальным значением\*/*

*Long j1j = i1i + 1;*

*loop {*

*if(m44s[j] < m44s[m1n]) m1n = j1j; fi*

*j1j++;*

*}*

*until j1j < n0n*

*/\* cделаем перестановку этого элемента, поменяв его местами с текущим\*/*

*if (i1i != m1n)*

*{*

*b8f = m44s[i1i];*

*m44s[i1i] = m44s[m1n];*

*m44s[m1n] = b8f;*

*}*

*i1i++;*

*}until( i1i < n0n)*

*/\* Конец сортировки\*/*

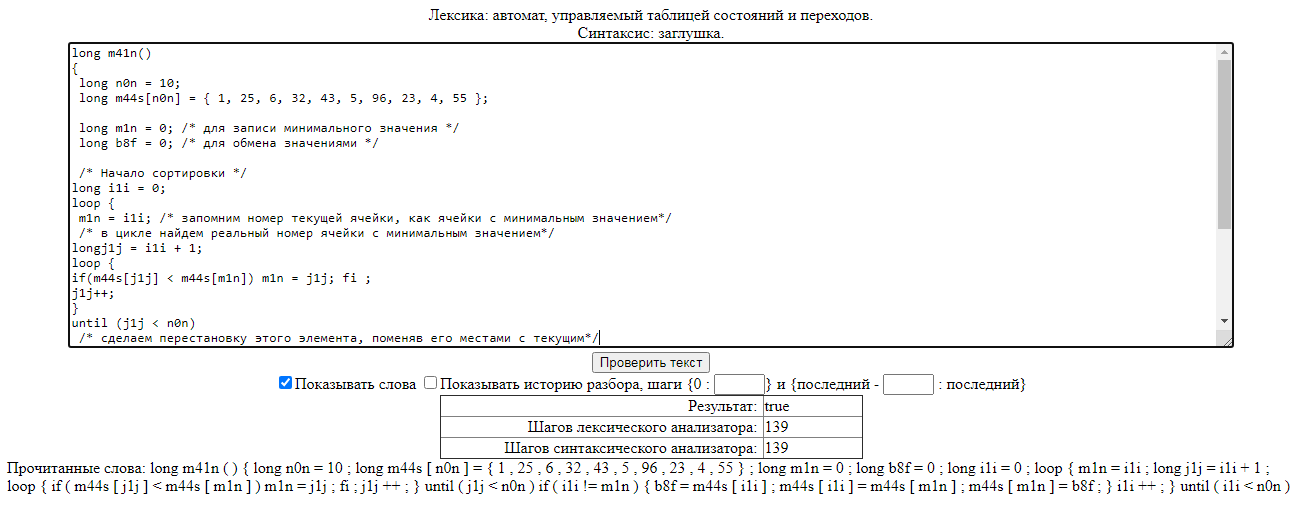


Рис.1 Проверка правильности написания лексики.

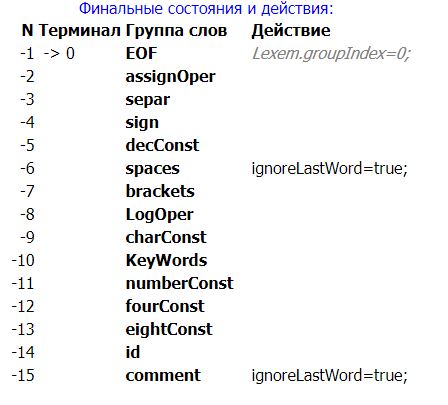


Рис.2 Финальные состояния и действия лексического анализатора

# Вывод

В ходе лабораторной работы были описаны основные свойства языка и разобраны основные операторы рассматриваемого языка. Также была разработана тестовая программа, по которой проверилась правильность написания лексики.

# Приложение

Текст xml-файла с системой правил:

1. **<?xml** version="1.0" encoding="windows-1251"**?>**
2. **<transLab>**
3. **<lexic>**
4. **<automat** name='main'**>**
5. **<rule** groupWordsName='KeyWords'**>**
6. **<expression>**("long""\_u"?)|("number")|("char")|("if")|("ifnot")|("fi")|("loop")|("until")|("switch")|("case")|("gout")|("void")|("return")**</expression>**
7. **</rule>**
8. **<rule** groupWordsName='LogOper'**>**
9. **<expression>**("++")|("--")|("**>**")|("**<**")|("**>**=")|("**<**=")|("||")|("&&")|("!=")|("==")|("!")**</expression>**
10. **</rule>**
11. **<rule** groupWordsName='brackets'**>**
12. **<expression>**[(){}[\]**<>**]**</expression>**
13. **</rule>**
14. **<rule** groupWordsName='eightConst'**>**
15. **<expression>**[8][xX][0-7]+**</expression>**
16. **</rule>**
17. **<rule** groupWordsName='id'**>**
18. **<expression>**[a-zA-Z][0-9]+[a-zA-Z]**</expression>**
19. **</rule>**
20. **<rule** groupWordsName='spaces'**>**
21. **<expression>**[ \n\r\t]+**</expression>**
22. **<action>**ignoreLastWord=true;**</action>**
23. **</rule>**
24. **<rule** groupWordsName='fourConst'**>**
25. **<expression>**[4][xX][0-3]+**</expression>**
26. **</rule>**
27. **<rule** groupWordsName='numberConst'**>**
28. **<expression>**[0-9]+[.][0-9]+**</expression>**
29. **</rule>**
30. **<rule** groupWordsName='decConst'**>**
31. **<expression>**[0] | ([1-9][0-9]\*)**</expression>**
32. **</rule>**
33. **<rule** groupWordsName='sign'**>**
34. **<expression>**[-+\*/]**</expression>**
35. **</rule>**
36. **<rule** groupWordsName='comment'**>**
37. **<expression>**[/][\*][]\*[\*][/]**</expression>**
38. **<action>**ignoreLastWord=true;**</action>**
39. **</rule>**
40. **<rule** groupWordsName='separ'**>**
41. **<expression>**[,:;]**</expression>**
42. **</rule>**
43. **<rule** groupWordsName='assignOper'**>**
44. **<expression>**[=]**</expression>**
45. **</rule>**
46. **<rule** groupWordsName='charConst'**>**
47. **<expression>**["'][]?["']**</expression>**
48. **</rule>**
49. **</automat>**
50. **</lexic>**
51. **</transLab>**