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*»**

|  |  |
| --- | --- |
| Выполнили: | Проверил: |
| Студент гр. АВТ-709, АВТФ | *к.т.н., доцент* |
| *Рассомахин И. А.* | *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: 22231313

**Описание варианта:**

1. Идентификатор:

<бБ><пЦ><бБ>

<одна любая буква><непустая последовательность цифр><одна любая буква>

Примеры: *d23U, N1q, x15y.*

1. Константы:

целые по основаниям 2,8 и 10 (*2x01, 8x77, 123*);

вещественные (*0.123, 123.321*);

символьные (*'g', 'a'*).

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

int[\_u]

float

letter

Примеры:

*int i9t* – целый;

*float f104t* – вещественный;

*letter l3113r* – символьный.

1. Оператор присваивания:

<И> := <В>;

<Идентификатор> := <произвольное выражение>;

Примеры:

*f104t := 123.321;*

*i9t := 12 + 34;*

1. Условный оператор

at <ЛВ> do <ОБ> [ else do <ОБ> ]

at <логическое выражение> do <оператор или блок> [else do <оператор или блок>]

Примеры:

# *at l0w < h1g do*

# *{*

# *int i9t := 0;*

*i9t := 23;*

# *}*

# *at i9t < 100 do i9t := 100;*

*else do i9t := 0;*

1. Оператор цикла:

cycle([<ОП>];[<ЛВ>];[<О>]) <ОБ>

cycle([<оператор присваивания>];[<логическое выражение>];[<одиночный оператор>]) <оператор или блок>

Примеры:

*cycle (;;) {}*

*cycle (i9t := 1; i9t < 100; i9t++) f104t := f104t + (i9t / 5);*

1. Опертаор переключателя:

select <В> case (<К> ) <ОБ> [break;] …[ case () <ОБ> ] end

select <произвольное выражение> case (<константа> ) <оператор или блок> [break;] …[ case () <оператор или блок> ] end

Пример:

*select i9t case (1) i9t :=0; break; case (2) i9t := 3; end*

*select i9t case (1) i9t :=0; break; case (2) i9t := 3; break; case () {*

*t3p := i9t;*

*i9t := i92t;*

*i92t := t3p;*

*}*

*end*

# 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:

*float p123r(float f104t)*

*{*

*return f104t \* f104t - f104t;*

*}*

Пример вызова вышеописанной функции:

*float 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

, 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  
/:= поделить равно

% знак остатка от деления

%:= остаток равно

+ плюс

+:= плюс равно

\* знак умножения

\*:= умножить равно

- минус

-:= минус равно

== равно

++ инкремент

- - декремент

**Комментарии**

Комментарии — это последовательность символов, игнорируемая компилятором.

Комментарии могут быть везде, где допустимо появление разделительных символов.

Пример:

*float f104t; //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:  
*int*

*int \_u*

*float*

*letter*

*void*

*return*

*cycle*

*select*

*case*

*break*

*end*

*at*

*do*

*else*

**Тестоваяпрограмма*:***

*//quick sort prog*

*// A utility function to swap two elements*

*void s84p(int a1a, int b2b)*

*{*

*int t3t := a1a;*

*a1a := b2b;*

*b2b := t3t;*

*}*

*// This function takes last element as pivot, places*

*//the pivot element at its correct position in sorted*

*//array, and places all smaller (smaller than pivot)*

*//to left of pivot and all greater elements to right*

*//of pivot*

*int p1n (int a4r [], int l0w, int h1g)*

*{*

*int p180t := a4r[h1g]; // pivot*

*int i1i := (l0w - 1); // Index of smaller element*

*cycle (int j1j := l0w; j1j <:= h1g - 1; j1j++)*

*{*

*// If current element is smaller than the pivot*

*at a4r[j1j] < p180t do*

*{*

*i1i++; // increment index of smaller element*

*s84p(a4r[i1i], a4r[j1j]);*

*}*

*}*

*s84p(a4r[i1i + 1], a4r[h1g]);*

*return (i1i + 1);*

*}*

*// The main function that implements QuickSort*

*//arr[] --> Array to be sorted,*

*//low --> Starting index,*

*//high --> Ending index \*/*

*void q1S(int a4r[], int l0w, int h1g)*

*{*

*at l0w < h1g do*

*{*

*// pi is partitioning index, arr[p] is now*

*// at right place*

*int p3i := p1n(a4r, l0w, h1g);*

*// Separately sort elements before*

*// partition and after partition*

*q1S(a4r, l0w, p3i - 1);*

*q1S(a4r, p3i + 1, h1g);*

*}*

*}*

*int m41n()*

*{*

*int a4r[] := {10, 7, 8, 9, 1, 5};*

*int n1n := s1530f(a4r) / s1530f(a4r[0]);*

*q1S(a4r, 0, n1n - 1);*

*return 0;*

*}*

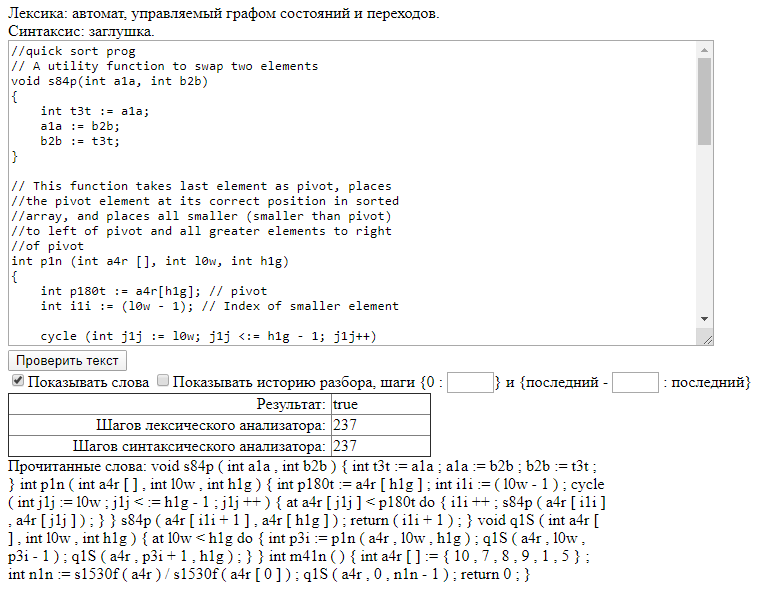


Рис.1 Проверка правильности написания лексики.

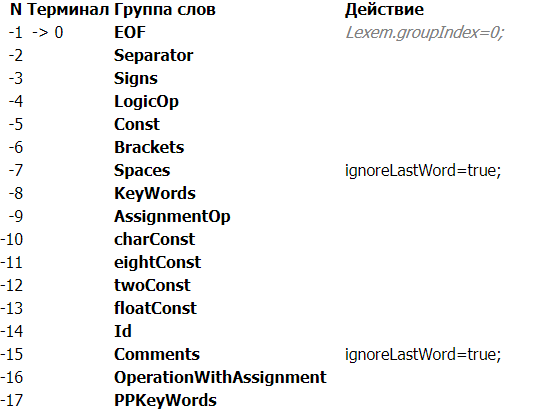


Рис.2 Финальные состояния и действия лексического анализатора

# 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='Spaces'**>**
6. **<expression>**[ \n\r\t]+**</expression>**
7. **<action>**ignoreLastWord=true;**</action>**
8. **<comment>**Разделители**</comment>**
9. **</rule>**
10. **<rule** groupWordsName='Brackets'**>**
11. **<expression>**[()[\]{}]**</expression>**
12. **<comment>**Скобки**</comment>**
13. **</rule>**
14. **<rule** groupWordsName='Signs'**>**
15. **<expression>**[-+\*%/]**</expression>**
16. **<comment>**Операторы математических операций**</comment>**
17. **</rule>**
18. **<rule** groupWordsName='OperationWithAssignment'**>**
19. **<expression>**[-+\*!%/][:][=]**</expression>**
20. **<comment>**Операторы математических операций с присваиванием**</comment>**
21. **</rule>**
22. **<rule** groupWordsName='Id'**>**
23. **<expression>**[a-zA-Z][0-9]+[a-zA-Z]**</expression>**
24. **<comment>**Идентификатор по варианту**</comment>**
25. **</rule>**
26. **<rule** groupWordsName='Const'**>**
27. **<expression>**([1-9][0-9]\*)|([0])**</expression>**
28. **<comment>**Десятичное целое число**</comment>**
29. **</rule>**
30. **<rule** groupWordsName='twoConst'**>**
31. **<expression>**[2][xX][0-1]\***</expression>**
32. **<comment>**Двоичное целое число**</comment>**
33. **</rule>**
34. **<rule** groupWordsName='eightConst'**>**
35. **<expression>**[8][xX][0-7]\***</expression>**
36. **<comment>**Восьмеричное целое число**</comment>**
37. **</rule>**
38. **<rule** groupWordsName='floatConst'**>**
39. **<expression>**[0-9]\*[.][0-9]+**</expression>**
40. **<comment>**Десятичное вещественное число**</comment>**
41. **</rule>**
42. **<rule** groupWordsName='charConst'**>**
43. **<expression>**['][]?[']**</expression>**
44. **<comment>**Символ**</comment>**
45. **</rule>**
46. **<rule** groupWordsName='AssignmentOp'**>**
47. **<expression>**[:][=]**</expression>**
48. **<comment>**Оператор присваивания**</comment>**
49. **</rule>**
50. **<rule** groupWordsName='LogicOp'**>**
51. **<expression>**("++")|("--")|("**>**")|("**<**")|("**>**=")|("**<**=")|("||")|("&&")|("!=")|("==")|("!")**</expression>**
52. **<comment>**Логические операции**</comment>**
53. **</rule>**
54. **<rule** groupWordsName='Comments'**>**
55. **<expression>**[/][/][]\*[\n]**</expression>**
56. **<action>**ignoreLastWord=true;**</action>**
57. **<comment>**Комментарии**</comment>**
58. **</rule>**
59. **<rule** groupWordsName='Separator'**>**
60. **<expression>**[,;]**</expression>**
61. **<comment>**Разделительные знаки**</comment>**
62. **</rule>**
63. **<rule** groupWordsName='KeyWords'**>**
64. **<expression>**("int""\_u"?)|("float")|("letter")|("at")|("do")|("else")|("cycle")|("select")|("case")|("break")|("end")|("return")|("void")|("const")**</expression>**
65. **<comment>**Ключевые слова**</comment>**
66. **</rule>**
67. **<rule** groupWordsName='PPKeyWords'**>**
68. **<expression>**[#]("include")|("define")**</expression>**
69. **<comment>**Ключевые слова препроцессора**</comment>**
70. **</rule>**
71. **</automat>**
72. **</lexic>**
73. **</transLab>**