МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

Лабораторная работа №2

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

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

Вариант № 11213131

Группа: АВТ-709

Студент: Антонов К. О.

Преподаватель: Малявко А.А.

Новосибирск, 2020 г.

Оглавление

[Цель работы 3](#_Toc35443701)

[Краткое описание языка регулярных выражений пакета ВебТрансЛаб 3](#_Toc35443702)

[Вариант задания 3](#_Toc35443703)

[Описание структуры программных модулей конечных автоматов, управляемых таблично и графом состояний и переходов 5](#_Toc35443704)

[Пример кода 6](#_Toc35443705)

[Описание языка 11213131 9](#_Toc35443706)

[1. Структура программы 9](#_Toc35443707)

[2. Типы данных 9](#_Toc35443708)

[3. Идентификаторы 10](#_Toc35443709)

[4. Константы 10](#_Toc35443710)

[5. Переменные 11](#_Toc35443711)

[6. Операторы 12](#_Toc35443712)

[7. Элементы языка 14](#_Toc35443713)

[Вывод 15](#_Toc35443714)

[Приложение 16](#_Toc35443715)

# **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.

# **Краткое описание языка регулярных выражений пакета ВебТрансЛаб**

* [<произвольный символ>]. Такое регулярное выражение порождает единственную цепочку, состоящую из этого символа: [a] → a;
* [<перечень символов]. Такое выражение порождает несколько цепочек, каждая из которых содержит в точности один символ из указанного перечня: [aA] → a, [aA] → A;
* [<диапазон символов>], например [0-9]. Это выражение можно считать сокращенной формой записи выражения вида [0123456789]. Оно порождает 10 цепочек, каждая из которых содержит единственную десятичную цифру.

Простым называется произвольное регулярное выражение, заключенное в круглые скобки. К первичным и простым РВ применяются следующие знаки операций, используемые для описания цепочек, длина которых больше единицы, и пустых цепочек.

Список операций:

1. «Пусто или в точности одно» (?)
2. «Одно или несколько» (+)
3. «Пусто, одно или несколько» (\*)
4. Конкатенация: [a][b] → ab
5. Выбор: [a] | [b] → [ab]

# **Вариант задания**

Согласно варианту задания на лабораторную работу, структура языка имеет следующий вид:

1. *Идентификаторы:*

|  |
| --- |
| **1** |
| <Б>\_<пбЦ>  (A\_, X\_12, W\_out…) |  |  |  |

1. *Константы:*

|  |
| --- |
| 1 |
| целые по основаниям 4, 10 и 16;  вещественные;  строковые |

1. *Объявления примитивных типов (целое, вещественное, символьное):*

|  |
| --- |
| **2** |
| **int[\_u]**  **float**  **letter** |

1. *Оператор присваивания:*

|  |
| --- |
| **1** |
| <И> **=** <В>; |

1. *Условный оператор:*

|  |
| --- |
| **3** |
| **when**<ЛВ> **then** <ОБ>**[ other**<ОБ> **]** |

1. *Оператор цикла:*

|  |
| --- |
| 1 |
| **loop** <ОБ> **until**<ЛВ> |

1. *Оператор переключателя:*

|  |
| --- |
| 3 |
| **switch**<В> **{** **by** <К> **do**<ОБ> **[ off; ]** …**[ any**<ОБ> **]** **}** |

1. *Формат псевдокода:*

|  |
| --- |
| **1** |
| триады <Код><Оп><Оп> |

Обозначения:

[...] – необязательная часть конструкции;

… – предшествующая часть конструкции повторяется произвольное количество раз;

< > – описание сокращений:

<Б>|<б>|<бБ>|<бЦ>|<Ц>|<пБ>|<пЦ>|<пБЦ> – одна большая буква | одна маленькая буква | одна любая буква | одна маленькая буква или одна цифра одна цифра | непустая последовательность букв | непустая последовательность цифр | возможно пустая посл-ть букв и/или цифр

<И> – Идентификатор (имя переменной / объекта); <К> – Константа;

<В> – произвольное Выражение; <ЛВ> – Логическое Выражение;

<ОБ> – Оператор или Блок; <О> – одиночный оператор; <ОП> – оператор присваивания;

<Код> – поле кода операции; <М> – поле метки; <Оп> – поле наименования операнда; <Р> – поле наименования результата

# **Описание структуры программных модулей конечных автоматов, управляемых таблично и графом состояний и переходов**

Конечный автомат, управляемый графом состояний и переходов

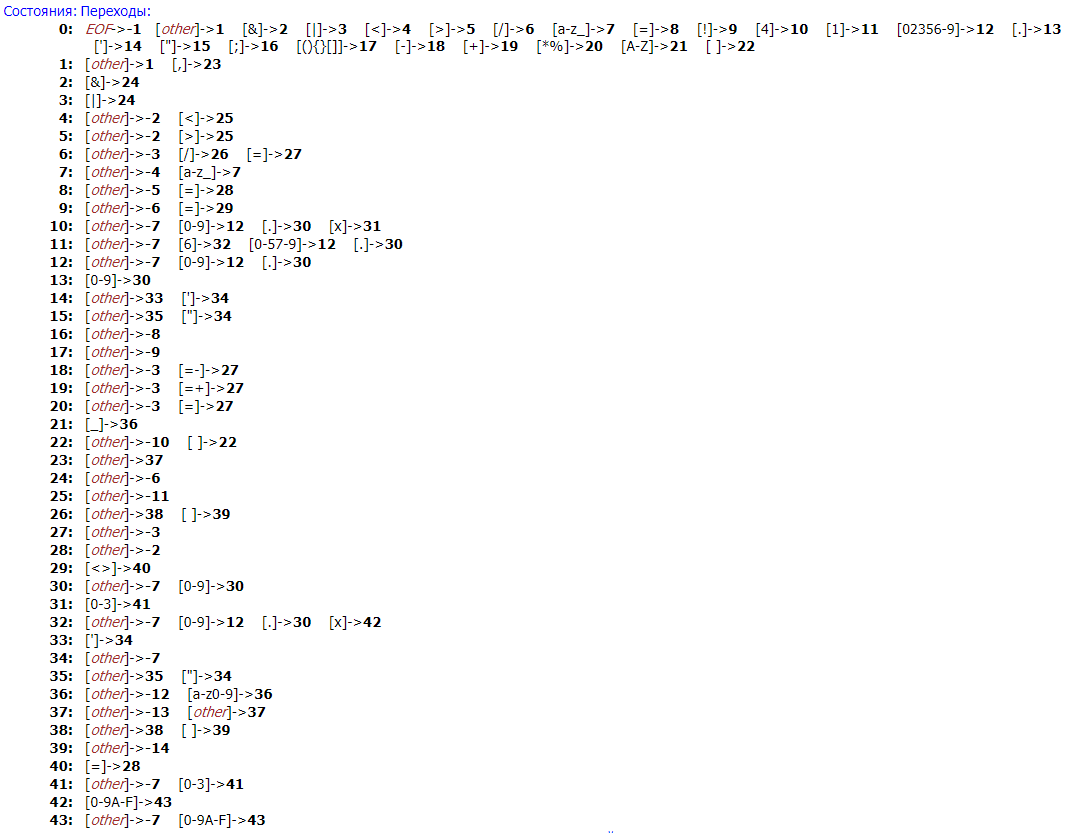


Рисунок 1 – Граф состояний и переходов лексического анализатора

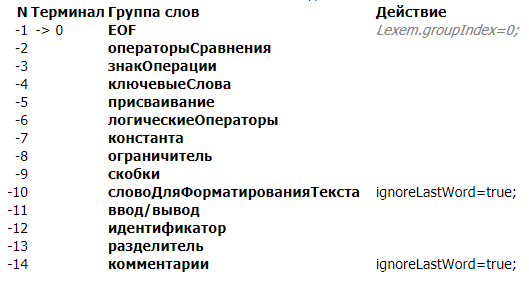


Рисунок 2 – Финальные состояния и действия

Данный граф описывает список состояний и возможных переходов между ними. Из состояния 0 можно перейти в одно из состояний 1-22, затем либо в рабочее, либо в финальное состояние.

Конечный автомат, управляемый таблично

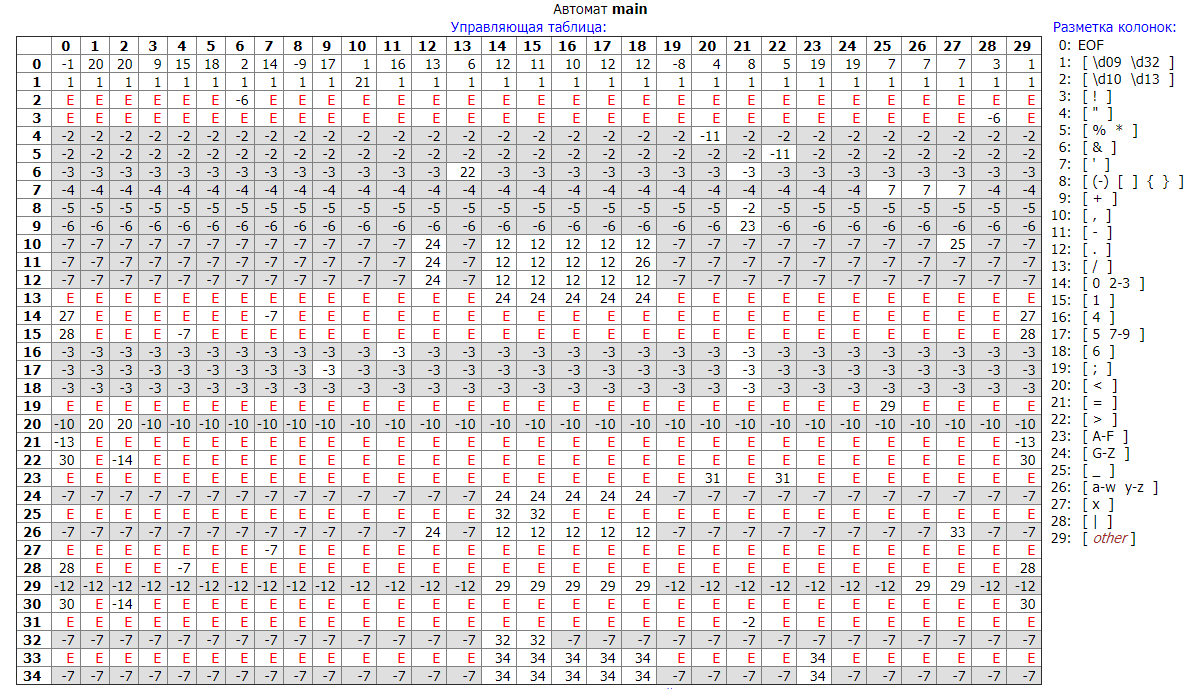


Рисунок 3 – Управляющая таблица

Обозначения в клетках управляющей таблицы:

* число > 0 – переход в рабочее состояние;
* число < 0 – переход в финальное состояние;
* число < 0 на сером фоне – переход в финальное состояние с возвратом литеры на вход;
* красное Е – останов по ошибке.

По данной таблице можно определить с каким результатом можно перейти в каждое из состояний.

# **Пример кода**

void F\_shellsort(int A\_[], int I\_size)

{

int I\_, J\_;

int I\_step = I\_size / 2;

loop

{

I\_ = 0;

loop

{

J\_ = I\_;

loop

{

int I\_temp = A\_[J\_];

A\_[J\_] = A\_[J\_ + I\_step];

A\_[J\_ + I\_step] = I\_temp;

J\_--;

}

until(J\_ >= 0 && A\_[J\_] > A\_[J\_ + I\_step])

I\_++;

}

until(I\_ < I\_size - I\_step)

I\_step /= 2;

}

until(I\_step > 0)

}

void F\_printarray(int A\_[], int I\_size)

{

int I\_ = 0;

loop

{

print(A\_[I\_]);

}

until(I\_ < I\_size)

out << "\n\n";

}

void main() {

const int I\_size = 10;

int I\_arr[size];

int I\_tmp = I\_size;

int I\_ = 0;

loop

{

I\_arr[I\_] = I\_tmp;

I\_tmp--;

I\_++;

}

until(I\_ < I\_size)

F\_printarray(I\_arr, I\_size);

F\_shellsort(I\_arr, I\_size);

F\_printarray(I\_arr, I\_size);

float A\_, B\_;

out << "Enter A\_: ";

in >> A\_;

out << "Enter B\_: ";

in >> B\_;

when (A\_ > B\_) then

out << "A\_ > B\_" << endl;

other when (A\_ == B\_) then

out << "A\_ == B\_" << endl;

other

out << "A\_ < B\_" << endl;

letter I\_switchVar;

out << "Enter I\_switchVar (c or d): ";

in >> I\_switchVar;

switch (I\_switchVar)

{

by 'c' do

out << "You have chosen 'c'" << endl;

off;

by 'd' do

out << "You have chosen 'd'" << endl;

off;

any

out << "Wrong variable" << endl;

off;

}

const int I\_size16 = 16xA;

int I\_4 = 4x10;

loop

{

out << I\_4;

I\_4++;

}

until(I\_4 < I\_size16)

}



Рисунок 4 – Правила лексики данного языка

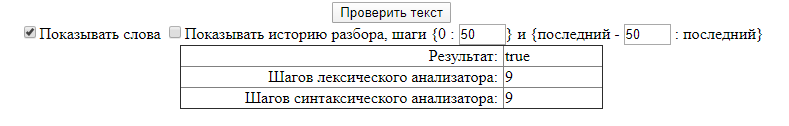


Рисунок 5 – Проверка лексики

# **Описание языка 11213131**

1. **Структура программы**

Программа на данном языке состоит из одной или более функций. Язык является блочно-структурированным. Каждый блок заключается в фигурные скобки. Основным блоком в программе является главная функция, которая записывается как void main().

Любая команда (вызов функции, осуществление некоторых операций) завершается разделителем – символом «точка с запятой».

Синтаксис объявления функции:

*<тип возвращаемого значения> <идентификатор> (<тип> <параметр1>, <тип> <параметр2>, …, <тип> <параметрN>)*

*{ <оператор или блок> [return [тип];] }*

Функция не может объявляться в других функциях. Функция может вызываться в другой функции, если первая была описана ранее.

1. **Типы данных**

Данный язык поддерживает статическую типизацию, то есть контроль типов осуществляется при компиляции.

Целочисленный тип данных определяется ключевым словом «int». К данному типу относятся числа по основанию 4, 10 и 16.

Вещественный тип данных определяется ключевым словом «float». К данному типу относятся числа, имеющие дробную часть. Дробная часть отделяется от целочисленной символом «.».

Символьный тип данных определяется ключевым словом «letter». К данному типу относятся любые одиночные символы. При присваивании символ записывается через одинарные кавычки.

В данном языке существуют массивы – структура данных с однотипным набором данных. Синтаксис определения:

*<тип> <идентификаторМассива> [<длинаМассива>]*

1. **Идентификаторы**

В данном языке идентификатором может быть последовательность букв латинского алфавита A-Za-z, цифр 0 – 9 и знак подчеркивания (\_). Первым символом всегда должна быть заглавная латинская буква, вторым – нижнее подчеркивание, остальные в любом порядке (могут и отсутствовать). Перед идентификатором должен быть указан тип данных.

1. **Константы**

В данном языке определены числовые и символьные константы. Числовые константы делятся на целочисленные константы и вещественные.

Целочисленные константы могут быть представлены в следующих системах счисления: 4, 10 и 16.

Десятичная константа записывается как последовательность цифр 0 – 9 без объявления системы счисления.

Целочисленные константы, имеющие четверичную и шестнадцатеричную систему счисления, начинаются с соответствующего значения их системы, и через символ «x» (лат.) указываются числа. Например, 4x21 или 16x07C.

Шестнадцатеричные константы в своем алфавите кроме чисел 0 – 9 имеют латинские буквы A – F.

Целая и дробная часть вещественной константы разделяются символом «.». При этом дробная часть может быть опущена при наличии целой, и наоборот. Например, 5., 0.5, .5.

Символьная константа – это один символ, записанный в одинарные кавычки. В качестве символьных констант также могут использоваться управляющие коды, не имеющие графического представления, при этом код управляющего символа начинается с символа «\».

Таблица 1 – Описание управляющих символов

|  |  |
| --- | --- |
| Обозначение | Описание |
| ‘\t’ | Табуляция |
| ‘\n’ | Перевод строки |
| ‘\r’ | Возврат каретки |

Строковые константы представляют собой любую последовательность, записанную между одиночных кавычек «’ ‘».

1. **Переменные**

Переменные могут быть локальными и глобальными. Локальные переменные существуют только в блоке кода, в котором они объявлены. Таким образом, локальные переменные создаются при входе в блок и уничтожаются при выходе из него. Глобальные переменные существует во всем файле программы.

Любая переменная до ее использования должна быть объявлена, то есть для неё должны быть указаны тип и идентификатор.

Объявление переменных осуществляется в следующей форме:

*<тип> <идентификатор>;*

Если в программе требуется несколько переменных одного типа, то они могут быть определены в одной строке через запятую:

*<тип> <идентикатор1>, <идентификатор2>;*

При объявлении переменной ей может быть присвоено начальное значение в форме:

*<тип> <идентификатор> = <значение>;*

1. **Операторы**

Над объектами могут выполняться следующие операции:

* операции присваивания;
* операции отношения;
* арифметические;
* логические.

**Оператор присваивания** имеет следующий синтаксис:

*<идентификатор> = <идентификатор/константа/выражение>;*

Операция присваивания выполняется в 2 этапа: сначала вычисляется выражение в правой части, затем результат присваивается операнду, стоящему в левой части.

Если объекты в правой и левой части относятся к разным типам, то используется операция явного приведения типа:

*<идентификатор> = (<тип>)<идентификатор/константа/выражение>;*

Определены следующие **операции отношения**:

* == – проверка на равенство;
* != – проверка на неравенство;
* < – меньше;
* > – больше;
* <= – меньше или равно;
* >= – больше или равно.

**Арифметические операции**, расположенные в порядке уменьшения приоритета:

Бинарные:

* \* – умножение;
* / – деление;
* + – сложение;
* - – вычитание;
* % – остаток от целочисленного деления.

Унарные:

* ++ – инкремент;
* -- – декремент;
* - – изменение знака.

Бинарные арифметические операции могут быть объединены с операцией присваивания:

*<объект> \*= <идентификатор/константа/выражение>;*

*<объект> /= <идентификатор/константа/выражение>;*

*<объект> += <идентификатор/константа/выражение>;*

*<объект> -= <идентификатор/константа/выражение>;*

*<объект> %= <идентификатор/константа/выражение>;*

**Логические операции** используются в операциях проверки условия «when» и могут выполняться над любыми объектами. Результат условной логической операции равен единице, если выражение истинно, иначе равен нулю.

Все значения, отличные от нуля, интерпретируются как истинные.

* && – И (бинарная) – оба выражения должны быть истинны;
* || – ИЛИ (бинарная) – одно из выражений должно быть истинно;
* ! – НЕ (унарная) – отрицание.

**Условный оператор «when»** имеет следующую структуру:

*when (<логическое выражение>) then*

*<оператор/блок>*

*[other when (<логическое выражение>) then*

*<оператор или блок>]*

*[other <оператор или блок>]*

Описание условного оператора начинается с ключевого слова «when», за которым идет логическое выражение. После ключевого слова «then» описывается блок действий, который будет выполняться при выполнении условия. Если нужно проверить несколько вариантов условия, то используется блок «other when» (аналог «else if»). Блок «other» выполняется при невыполнении вышестоящих условий.

**Оператор цикла «loop»** позволяет выполняет одно и то же действие несколько раз подряд.

*loop <оператор/блок> until (<логическое выражение>)*

Данный цикл будет выполняться, пока логическое выражение является истиной.

**Оператор перехода «switch»** – это удобная замена длинной «when-other» конструкции, которая сравнивает переменную с несколькими константными значениями, например «int» или «letter». Синтаксис:

*switch (<переменная>) {*

*by <значение1> do*

*<оператор/блок>*

*off;*

*by <значение2> do*

*<оператор/блок>*

*off;*

*…*

*any*

*<оператор/блок>*

*off;*

*}*

Переменная в скобках сравнивается со значениями, описанными после ключевого слова «by». После ключевого слова «do» находится код, который будет выполнен в случае, если переменная оказалась равной текущему значению. Ключевое слово «off» необходимо для того, чтобы прервать выполнение блока. Блок «any» необязателен, но он полезен для обработки исключительных ситуаций.

1. **Элементы языка**

Алфавит данного языка программирования включает:

* прописные и строчные латинские буквы, а также знак подчеркивания;
* арабские цифры 0-9;
* специальные знаки: " {}, | [] () + - / % \* . \ < = > ! & ; '
* пробельные символы: пробел, символ табуляции, символ перехода на новую строку.

Из символов алфавита формируются лексемы языка (минимальная единица языка, имеющая самостоятельный смысл):

* идентификаторы;
* ключевые слова;
* знаки операций;
* константы;
* разделители.

Список ключевых слов языка:

|  |  |  |  |
| --- | --- | --- | --- |
| int | const | loop | do |
| float | when | until | off |
| letter | then | switch | any |
| void | other | by | return |

В данном языке используются однострочные комментарии с помощью последовательности символов «//». Код, расположенный после этих символов, не будет компилироваться до конца строки.

# **Вывод**

Изучили конечные автоматы без памяти, способы определения КА, методы построения недетерминированного КА по системе регулярных выражений, а также описали язык программирования 11213131 в соответствии с вариантом.

# **Приложение**

Приложение 1. Текст автомата, реализованного при помощи графа

1. //Построено 18.3.2020 17:01:54 по исходному файлу/шаблону 1lab.xml/lexAsGraphSyntAsSxLR\_to\_js
2. var ERROR\_VALUE = false;
3. var lCnt = 0;
4. var cCnt = 0;
5. var stk = [];
6. //входной поток литер
7. var TextReader = {
8. text: "^",
9. len: -1,
10. backedChar: 0,
11. pos: 0,
13. setText: function(t){
14. this.text = t;
15. this.len = t.length - 1;
16. this.pos = 0;
17. },
18. //возврат заданной литеры во входной поток
19. back: function(bc){
20. if(this.pos > 0){    //если нет возвращенной, но не прочитанной литеры
21. this.pos = -this.pos;   //устанавливаем признак наличия возвращенной литеры
22. this.backedChar = bc;       //и запоминаем ее код
23. }
24. },
25. //чтение одной литеры
26. read: function() {
27. if(this.pos >= 0){   //если нет возвращенной литеры
28. //и возвращаем -1, если достигнут конец входного потока (или входного потока просто нет), или очередную литеру из него
29. return this.pos <= this.len? this.text[this.pos++]:null;
30. }
31. else    //если есть возвращенная литера
32. {
33. this.pos = -this.pos;   //сбросим признак ее наличия
34. return this.backedChar; //и вернем ее вновь
35. }
36. }
37. };
39. // лексический акцептор
40. //вспомогательные классы
42. //вершина графа состояний и переходов - состояние конечного автомата
43. function FAstate(toCount){
44. this.count = toCount;
45. this.markTo = [];   //создание массива дуг
46. this.statesTo = []; //создание массива номеров вершин
47. this.filled=0;  //установка нулевой степени заполненности массивов
48. }
49. FAstate.prototype = {
50. //добавление дуги, ведущей в состояние to с разметкой mark
51. setArc: function(to, mark){
52. if(this.filled < this.count){    //если в массивах есть место для новой дуги
53. this.statesTo[this.filled] = to;    //запомним, куда она ведет
54. this.markTo[this.filled++] = mark;
55. }
56. },
57. //определить, ведет ли в какое-нибудь состояние дуга по заданной литере
58. getState: function(c) {
59. var i;
60. var forAll = 0;
61. for(i = 0; i < this.filled; i++){    //просмотреть все дуги
62. if(this.markTo[i] == "")
63. forAll = this.statesTo[i];
64. if(this.markTo[i].indexOf(c) >= 0)
65. return this.statesTo[i];
66. }
67. return forAll == 0? null:forAll;
68. },
69. //вернуть номер финального состояния, если в такое состояние ведет хотя бы одна дуга
70. getFinalState: function(){
71. var i;
72. for(i = 0; i < this.filled; i++){    //просмотреть все дуги
73. if(this.statesTo[i] < 0) //если дуга ведет в финальное состояние
74. return this.statesTo[i];    // вернуть его номер
75. }
76. return null;    //если нет дуг в финальные состояния - вернуть максимально возможное отрицательное число
77. }
78. };
80. //лексический акцептор, управляемый графом состояний и переходов
81. //конструктор
82. function Fautomat(no, ind, inp){
83. this.index = no;
84. this.reader = inp;
85. this.states = [];
86. }
87. Fautomat.prototype = {
88. //сохранение заданного состояния (вершины)
89. setState: function(no, state){
90. this.states[no] = state;    //сохраним состояние в массиве
91. },
93. //основной метод лексического акцептора - чтение одного слова из входного потока и формирование лексемы
94. getLexem: function(){
95. var curState;   //текущее состояние (-1 на случай, если входной поток ранее был исчерпан)
96. var curChar;    //текущая литеа (-1 на случай, если входной поток ранее был исчерпан)
97. var newState;
98. var word = "";
100. for(curState = 0; curState>=0; ){    //запускаем цикл имитации работы конечного автомата, выполняющийся до тех пор, пока не было перехода в финальное состояние или не обнаружена ошибка
101. curChar = this.reader.read();   //пытаемся получить очередную литеру
102. if(curChar === null) {  //если нет больше литер в тексте
103. //формируем текущее состояние
104. //-1, если конец входного потока обнаружен в момент запуска автомата
105. //и номер финального состояния, в которое из текущего есть переход по эпсилон в противном случае
106. //(возможен случай, когда такого перехода нет, тогда получим значение -2147483647)
107. curState = (curState == 0 ? -1 : this.states[curState].getFinalState());
108. break;
109. } else {    //если код литеры получен
110. newState = this.states[curState].getState(curChar); //получаем новый номер состояния
111. if(newState === null)
112. curState = this.states[curState].getFinalState();
113. else
114. curState = newState;
115. //если номер нового состояния вне пределов массива состояний или код литеры меньше нуля - выйдем из цикла
116. if((curState < 0) || (curState === null) || (curState >= this.states.length))
117. break;
118. word += curChar;    //если по этой литере был выполнен переход в рабочее состояние - добавим ее к тексту слова
119. }
120. }
121. if(curState < -1)    //если не EndOfFile и не ошибка (в частности ошибкой является случай, когда curState>=0)
122. this.reader.back(curChar);  //вернем литеру во входной поток
124. return [curState, word];    //вернем лексему
125. },
126. //возвращает индекс акцептора, нужен в мультиавтоматном варианте
127. getIndex: function() {  //возвращает индекс акцептора, нужен в мультиавтоматном варианте
128. return this.index;
129. }
130. };
132. // Лексический анализатор


136. //одноавтоматный ЛА
137. function LexAnalyzer(rdr) {
138. var state;
139. //переменная для временного хранения состояния конечного автомата
140. //дальше идет создание автомата, его состояний, выходящих из них дуг и запоминание всего этого в соответствующих местах
141. this.lexAcceptor = new Fautomat(0,44,rdr);
142. state = new FAstate(22);
143. state.setArc(1,"");
144. state.setArc(2,"&");
145. state.setArc(3,"|");
146. state.setArc(4,"<");
147. state.setArc(5,">");
148. state.setArc(6,"/");
149. state.setArc(7,"abcdefghijklmnopqrstuvwxyz\_");
150. state.setArc(8,"=");
151. state.setArc(9,"!");
152. state.setArc(10,"4");
153. state.setArc(11,"1");
154. state.setArc(12,"02356789");
155. state.setArc(13,".");
156. state.setArc(14,"\'");
157. state.setArc(15,"\"");
158. state.setArc(16,";");
159. state.setArc(17,"(){}[]");
160. state.setArc(18,"-");
161. state.setArc(19,"+");
162. state.setArc(20,"\*%");
163. state.setArc(21,"ABCDEFGHIJKLMNOPQRSTUVWXYZ");
164. state.setArc(22," \u0009\r\n");
165. this.lexAcceptor.setState(0,state);
166. state = new FAstate(2);
167. state.setArc(1,"");
168. state.setArc(23,",");
169. this.lexAcceptor.setState(1,state);
170. state = new FAstate(1);
171. state.setArc(24,"&");
172. this.lexAcceptor.setState(2,state);
173. state = new FAstate(1);
174. state.setArc(24,"|");
175. this.lexAcceptor.setState(3,state);
176. state = new FAstate(2);
177. state.setArc(-2,"");
178. state.setArc(25,"<");
179. this.lexAcceptor.setState(4,state);
180. state = new FAstate(2);
181. state.setArc(-2,"");
182. state.setArc(25,">");
183. this.lexAcceptor.setState(5,state);
184. state = new FAstate(3);
185. state.setArc(-3,"");
186. state.setArc(26,"/");
187. state.setArc(27,"=");
188. this.lexAcceptor.setState(6,state);
189. state = new FAstate(2);
190. state.setArc(-4,"");
191. state.setArc(7,"abcdefghijklmnopqrstuvwxyz\_");
192. this.lexAcceptor.setState(7,state);
193. state = new FAstate(2);
194. state.setArc(-5,"");
195. state.setArc(28,"=");
196. this.lexAcceptor.setState(8,state);
197. state = new FAstate(2);
198. state.setArc(-6,"");
199. state.setArc(29,"=");
200. this.lexAcceptor.setState(9,state);
201. state = new FAstate(4);
202. state.setArc(-7,"");
203. state.setArc(12,"0123456789");
204. state.setArc(30,".");
205. state.setArc(31,"x");
206. this.lexAcceptor.setState(10,state);
207. state = new FAstate(4);
208. state.setArc(-7,"");
209. state.setArc(32,"6");
210. state.setArc(12,"012345789");
211. state.setArc(30,".");
212. this.lexAcceptor.setState(11,state);
213. state = new FAstate(3);
214. state.setArc(-7,"");
215. state.setArc(12,"0123456789");
216. state.setArc(30,".");
217. this.lexAcceptor.setState(12,state);
218. state = new FAstate(1);
219. state.setArc(30,"0123456789");
220. this.lexAcceptor.setState(13,state);
221. state = new FAstate(2);
222. state.setArc(33,"");
223. state.setArc(34,"\'");
224. this.lexAcceptor.setState(14,state);
225. state = new FAstate(2);
226. state.setArc(35,"");
227. state.setArc(34,"\"");
228. this.lexAcceptor.setState(15,state);
229. state = new FAstate(1);
230. state.setArc(-8,"");
231. this.lexAcceptor.setState(16,state);
232. state = new FAstate(1);
233. state.setArc(-9,"");
234. this.lexAcceptor.setState(17,state);
235. state = new FAstate(2);
236. state.setArc(-3,"");
237. state.setArc(27,"=-");
238. this.lexAcceptor.setState(18,state);
239. state = new FAstate(2);
240. state.setArc(-3,"");
241. state.setArc(27,"=+");
242. this.lexAcceptor.setState(19,state);
243. state = new FAstate(2);
244. state.setArc(-3,"");
245. state.setArc(27,"=");
246. this.lexAcceptor.setState(20,state);
247. state = new FAstate(1);
248. state.setArc(36,"\_");
249. this.lexAcceptor.setState(21,state);
250. state = new FAstate(2);
251. state.setArc(-10,"");
252. state.setArc(22," \u0009\r\n");
253. this.lexAcceptor.setState(22,state);
254. state = new FAstate(1);
255. state.setArc(37,"");
256. this.lexAcceptor.setState(23,state);
257. state = new FAstate(1);
258. state.setArc(-6,"");
259. this.lexAcceptor.setState(24,state);
260. state = new FAstate(1);
261. state.setArc(-11,"");
262. this.lexAcceptor.setState(25,state);
263. state = new FAstate(2);
264. state.setArc(38,"");
265. state.setArc(39,"\r\n");
266. this.lexAcceptor.setState(26,state);
267. state = new FAstate(1);
268. state.setArc(-3,"");
269. this.lexAcceptor.setState(27,state);
270. state = new FAstate(1);
271. state.setArc(-2,"");
272. this.lexAcceptor.setState(28,state);
273. state = new FAstate(1);
274. state.setArc(40,"<>");
275. this.lexAcceptor.setState(29,state);
276. state = new FAstate(2);
277. state.setArc(-7,"");
278. state.setArc(30,"0123456789");
279. this.lexAcceptor.setState(30,state);
280. state = new FAstate(1);
281. state.setArc(41,"0123");
282. this.lexAcceptor.setState(31,state);
283. state = new FAstate(4);
284. state.setArc(-7,"");
285. state.setArc(12,"0123456789");
286. state.setArc(30,".");
287. state.setArc(42,"x");
288. this.lexAcceptor.setState(32,state);
289. state = new FAstate(1);
290. state.setArc(34,"\'");
291. this.lexAcceptor.setState(33,state);
292. state = new FAstate(1);
293. state.setArc(-7,"");
294. this.lexAcceptor.setState(34,state);
295. state = new FAstate(2);
296. state.setArc(35,"");
297. state.setArc(34,"\"");
298. this.lexAcceptor.setState(35,state);
299. state = new FAstate(2);
300. state.setArc(-12,"");
301. state.setArc(36,"abcdefghijklmnopqrstuvwxyz0123456789");
302. this.lexAcceptor.setState(36,state);
303. state = new FAstate(2);
304. state.setArc(-13,"");
305. state.setArc(37,"");
306. this.lexAcceptor.setState(37,state);
307. state = new FAstate(2);
308. state.setArc(38,"");
309. state.setArc(39,"\r\n");
310. this.lexAcceptor.setState(38,state);
311. state = new FAstate(1);
312. state.setArc(-14,"");
313. this.lexAcceptor.setState(39,state);
314. state = new FAstate(1);
315. state.setArc(28,"=");
316. this.lexAcceptor.setState(40,state);
317. state = new FAstate(2);
318. state.setArc(-7,"");
319. state.setArc(41,"0123");
320. this.lexAcceptor.setState(41,state);
321. state = new FAstate(1);
322. state.setArc(43,"0123456789ABCDEF");
323. this.lexAcceptor.setState(42,state);
324. state = new FAstate(2);
325. state.setArc(-7,"");
326. state.setArc(43,"0123456789ABCDEF");
327. this.lexAcceptor.setState(43,state);
328. };
329. // распознавание слова и формирование лексемы
330. LexAnalyzer.prototype = {
331. getAcceptor: function(){
332. return this.lexAcceptor;
333. },
334. getLexem: function() {
335. var ignoreLastWord = true;  //временно установим флажок, чтобы войти в цикл
336. var lexem;
337. while(ignoreLastWord){  //пока флажок установлен - читаем очередное слово
338. ignoreLastWord = false; //перед чтением следующего слова сбросим флажок
339. //вызовем лексический акцептор и получим от него лексему:
340. lexem = this.lexAcceptor.getLexem();
341. //для каждой группы слов обеспечим возможность выполнения действий (имена групп слов в примечаниях)
342. //индексы групп слов, имена которых используются в грамматике, преобразуются в те значения, который присвоил им построитель синтаксического акцептора
343. //в действиях, в частности, может быть взведен флажок ignoreLastWord и изменены поля лексемы, установленные акцептором
344. switch(lexem[0]){
345. case -1: //EOF
346. lexem[0] = 0;
347. break;
348. case -2: //операторыСравнения
349. break;
350. case -3: //знакОперации
352. break;
353. case -4: //ключевыеСлова
354. break;
355. case -5: //присваивание
356. break;
357. case -6: //логическиеОператоры
358. break;
359. case -7: //константа




365. break;
366. case -8: //ограничитель
367. break;
368. case -9: //скобки
369. break;
370. case -10: //словоДляФорматированияТекста
371. ignoreLastWord=true;break;
372. case -11: //ввод/вывод
374. break;
375. case -12: //идентификатор
376. break;
377. case -13: //разделитель
378. break;
379. case -14: //комментарии
380. ignoreLastWord=true;break;
381. }
382. }
383. lCnt += 1;
384. return lexem;
385. }
386. };

389. var la;
390. var words;
391. var wList;
392. var stk = [];
393. var currentLexem = [];
394. var currentWord;
395. var curWordIndex;
396. var stackPush = function(val){
397. stk.push(val);
398. };
399. var stackPop = function(cnt){
400. for(;cnt>0;cnt-=1)
401. if(stk.length > 0)
402. stk.pop();
403. else
404. return true;
405. return false;
406. };
407. var stateFromStack = function(){
408. return ((stk.length <= 0)? -1 :stk[stk.length-1]);
409. };
410. // синтаксический акцептор/анализатор
411. function Parser(rdr){
412. stk = [];
413. currentLexem = [];
414. la = new LexAnalyzer(rdr);  //экземпляр лексического анализатора
415. wList = [];
416. this.pHist = [];
417. this.cCnt = 0;  //начальное значение счетчика циклов
419. // заглушка для случая, когда нет синтаксических правил
420. };
421. Parser.prototype = {
422. getWordIndex: function(){
423. currentLexem = la.getLexem();
424. this.wList.push(currentLexem[1]);
425. return currentLexem[0];
426. },
427. getInfo: function(i){
428. switch (i) {
429. case 0:
430. return lCnt;
431. case 1:
432. return this.cCnt;
433. case 2:
434. return this.wList;
435. case 3:
436. return [];
437. }
438. return null;
439. },
440. parse: function(){
441. var wi;
442. lCnt = 0;
443. this.cCnt = 1;
444. this.wList = [];
445. while(((wi = this.getWordIndex()) != 0) && (wi != null) && (wi > -1000)){
446. if(this.cCnt++>998)
447. break;
448. }
449. return (wi == 0);
450. }
451. };
452. (function(){
453. document.getElementById("lexName").innerText = " автомат, управляемый графом состояний и переходов.";
454. document.getElementById("syntName").innerText = " заглушка.";
455. })();

Приложение 2. Текст автомата, реализованного при помощи управляющей таблицы

1. //Построено 18.3.2020 17:04:18 по исходному файлу/шаблону 1lab.xml/lexAsTableSyntAsRD\_to\_js
2. var lCnt = 0;
3. var cCnt = 0;
5. //входной поток литер
6. var TextReader = {
7. text: "^",
8. len: -1,
9. backedChar: 0,
10. pos: 0,
12. setText: function(t){
13. this.text = t;
14. this.len = t.length - 1;
15. this.pos = 0;
16. },
17. //возврат заданной литеры во входной поток
18. back: function(bc){
19. if(this.pos > 0){    //если нет возвращенной, но не прочитанной литеры
20. this.pos = -this.pos;   //устанавливаем признак наличия возвращенной литеры
21. this.backedChar = bc;       //и запоминаем ее код
22. }
23. },
24. //чтение одной литеры
25. read: function() {
26. if(this.pos >= 0){   //если нет возвращенной литеры
27. //и возвращаем -1, если достигнут конец входного потока (или входного потока просто нет), или очередную литеру из него
28. return this.pos <= this.len? this.text[this.pos++]:null;
29. }
30. else    //если есть возвращенная литера
31. {
32. this.pos = -this.pos;   //сбросим признак ее наличия
33. return this.backedChar; //и вернем ее вновь
34. }
35. }
36. };
38. // лексический акцептор
39. /\*\* Преобразователь кода символа в индекс.
40. \* Использует собственный вектор диапазонов
41. \* (первый символ, последний символ, индекс, счетчик преобразований)
42. \* Диапазоны могут добавляться и удаляться на лету.
43. \* При преобразовании диапазоны могут переупорядочиваться согласно частоте появления символов.
44. \* В результате переупорядочивания диапазоны, содержащие часто появляющиеся символы,
45. \* оказываются в начале вектора, а те, символы которых появляются редко, вытесняются в его конец
46. \*/
47. function CharToIndex() {
48. this.count = 0;
49. this.fChar = [];
50. this.lChar = [];
51. this.index = [];
52. //  charCnt = new int[limit];
53. //  byChar = new int[limit];
54. };
56. /\*\*Создается новый диапазон символов.
57. \* @param fChr первый символ диапазона
58. \* @param lChr последний символ диапазона
59. \* @param indexChr индекс, приписываемый диапазону
60. \*/
61. CharToIndex.prototype = {
62. addRange: function(fChr, lChr, indexChr){
63. //запись параметров диапазона в первое свободное место
64. this.fChar[this.count] = fChr;
65. this.lChar[this.count] = lChr;
66. this.index[this.count++] = indexChr;
67. },
69. getIndexOfChar: function(Chr) {
70. var i;
71. for(i = 0; i < this.count; i++)
72. if((Chr >= this.fChar[i]) && (Chr <= this.lChar[i]))  //символ больше/равен началу диапазона?
73. return this.index[i];   //вернем индекс
74. return -1;  //возращается, ели не найден диапазон.
75. }
76. };
77. //конструктор
78. function fAutomatAsTable(no, cToI, ctlTbl, tr){
79. this.index = no;    //запомним собственный индекс
80. this.charToI = cToI;
81. this.controlTable = ctlTbl;
82. this.inp = tr;
83. this.w = "";
84. }
85. fAutomatAsTable.prototype = {
87. //основной метод лексического акцептора - чтение одного слова из входного потока и формирование лексемы
88. getLexem: function(){
89. var curState = 0;
90. var newState = 0;
91. var curChar = -1;
92. var colIndex;   //объявление и инициализация локальных переменных
93. this.w = "";
94. for( ; ; ) {    //запускаем цикл имитации работы конечного автомата, выполняющийся до тех пор, пока не было перехода в финальное состояние или не обнаружена ошибка
95. if((curChar = this.inp.read()) === null)    //читаем очередную входную литеру; если возвращен признак исчерпания входного потока (-1)
96. colIndex = 0;   //то устанавливаем колонку УТ, соответствующую эпсилон
97. else
98. colIndex = this.charToI.getIndexOfChar(curChar);    //преобразуем код литеры в индекс колонки УТ
99. if((colIndex < 0) || (colIndex >= this.controlTable[curState].length))    //если индекс колонки находится за пределами УТ
100. //одноавтоматный вариант
101. if(curState == 0) {
102. curState = -2147483647;
103. break;
104. } else
105. colIndex = 0;
107. newState = this.controlTable[curState][colIndex];   //выбираем из УТ номер следующего состояния
108. if((newState == -2147483647) && (this.controlTable[curState][0] > 0))
109. newState = this.controlTable[curState][0];
110. curState = newState;
111. if((curChar === null) || (curState < 0)) //если исчерпан входной поток или автомат перходит в финальное состояние или обнаружена ошибка
112. break;  //выходим из цикла
113. this.w += curChar;  //добавляем литеру к тексту слова
114. }
115. if((curState & 0x40000000) != 0)
116. this.inp.back(curChar);
117. else
118. this.w += curChar;  //добавляем литеру к тексту слова
119. return [(curState >= 0 ? -2147483647 : curState | 0x40000000), this.w];  //вернем лексему
120. },
122. getIndex: function() {
123. return this.index;
124. }
125. };
127. // лексический анализатор
129. // данные из правил

132. function LexAnalyzer(rdr) {
133. this.controlTable = [];
134. this.charToI = new CharToIndex();
136. this.controlTable.push([-1,20,20,9,15,18,2,14,-1073741833,17,1,16,13,6,12,11,10,12,12,-1073741832,4,8,5,19,19,7,7,7,3,1,0]);
137. this.controlTable.push([1,1,1,1,1,1,1,1,1,1,21,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0]);
138. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741830,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
139. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741830,-2147483647,0]);
140. this.controlTable.push([-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-1073741835,-2,-2,-2,-2,-2,-2,-2,-2,-2,0]);
141. this.controlTable.push([-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-1073741835,-2,-2,-2,-2,-2,-2,-2,0]);
142. this.controlTable.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,22,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,0]);
143. this.controlTable.push([-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,7,7,7,-4,-4,0]);
144. this.controlTable.push([-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-5,-1073741826,-5,-5,-5,-5,-5,-5,-5,-5,0]);
145. this.controlTable.push([-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,-6,23,-6,-6,-6,-6,-6,-6,-6,-6,0]);
146. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,24,-7,12,12,12,12,12,-7,-7,-7,-7,-7,-7,-7,-7,25,-7,-7,0]);
147. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,24,-7,12,12,12,12,26,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,0]);
148. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,24,-7,12,12,12,12,12,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,0]);
149. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,24,24,24,24,24,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
150. this.controlTable.push([27,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741831,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,27,0]);
151. this.controlTable.push([28,-2147483647,-2147483647,-2147483647,-1073741831,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,28,0]);
152. this.controlTable.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,0]);
153. this.controlTable.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,0]);
154. this.controlTable.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,-3,0]);
155. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,29,-2147483647,-2147483647,-2147483647,-2147483647,0]);
156. this.controlTable.push([-10,20,20,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,-10,0]);
157. this.controlTable.push([-1073741837,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741837,0]);
158. this.controlTable.push([30,-2147483647,-1073741838,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,30,0]);
159. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,31,-2147483647,31,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
160. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,24,24,24,24,24,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,0]);
161. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,32,32,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
162. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,24,-7,12,12,12,12,12,-7,-7,-7,-7,-7,-7,-7,-7,33,-7,-7,0]);
163. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741831,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
164. this.controlTable.push([28,-2147483647,-2147483647,-2147483647,-1073741831,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,28,0]);
165. this.controlTable.push([-12,-12,-12,-12,-12,-12,-12,-12,-12,-12,-12,-12,-12,-12,29,29,29,29,29,-12,-12,-12,-12,-12,-12,-12,29,29,-12,-12,0]);
166. this.controlTable.push([30,-2147483647,-1073741838,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,30,0]);
167. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741826,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
168. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,32,32,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,0]);
169. this.controlTable.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,34,34,34,34,34,-2147483647,-2147483647,-2147483647,-2147483647,34,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);
170. this.controlTable.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,34,34,34,34,34,-7,-7,-7,-7,34,-7,-7,-7,-7,-7,-7,0]);
172. //транслитератор автомата main
173. this.charToI.addRange('\u0009','\u0009',1);
174. this.charToI.addRange(' ',' ',1);
175. this.charToI.addRange('\n','\n',2);
176. this.charToI.addRange('\r','\r',2);
177. this.charToI.addRange('!','!',3);
178. this.charToI.addRange('\"','\"',4);
179. this.charToI.addRange('%','%',5);
180. this.charToI.addRange('\*','\*',5);
181. this.charToI.addRange('&','&',6);
182. this.charToI.addRange('\'','\'',7);
183. this.charToI.addRange('(',')',8);
184. this.charToI.addRange('[','[',8);
185. this.charToI.addRange(']',']',8);
186. this.charToI.addRange('{','{',8);
187. this.charToI.addRange('}','}',8);
188. this.charToI.addRange('+','+',9);
189. this.charToI.addRange(',',',',10);
190. this.charToI.addRange('-','-',11);
191. this.charToI.addRange('.','.',12);
192. this.charToI.addRange('/','/',13);
193. this.charToI.addRange('0','0',14);
194. this.charToI.addRange('2','3',14);
195. this.charToI.addRange('1','1',15);
196. this.charToI.addRange('4','4',16);
197. this.charToI.addRange('5','5',17);
198. this.charToI.addRange('7','9',17);
199. this.charToI.addRange('6','6',18);
200. this.charToI.addRange(';',';',19);
201. this.charToI.addRange('<','<',20);
202. this.charToI.addRange('=','=',21);
203. this.charToI.addRange('>','>',22);
204. this.charToI.addRange('A','F',23);
205. this.charToI.addRange('G','Z',24);
206. this.charToI.addRange('\_','\_',25);
207. this.charToI.addRange('a','w',26);
208. this.charToI.addRange('y','z',26);
209. this.charToI.addRange('x','x',27);
210. this.charToI.addRange('|','|',28);
211. this.lexAcceptor = new fAutomatAsTable(0,this.charToI,this.controlTable,rdr);
213. }
214. // распознавание слова и формирование лексемы
215. LexAnalyzer.prototype = {
216. getLexem: function(){
217. ignoreLastWord = true;  //для входа в основной цикл установим флажок игнорирования слова
218. while(ignoreLastWord) {
219. ignoreLastWord = false; //и сбросим его (возможно, он будет установлен действием)
220. //вызов лексического акцептора:
221. lexem = this.lexAcceptor.getLexem();    //получим лексему от текущего лексического акцептора
222. //для каждой группы слов обеспечим возможность выполнения действий (имена групп слов в примечаниях)
223. //индексы групп слов, имена которых используются в грамматике, преобразуются в те значения, который присвоил им построитель синтаксического акцептора
224. //в действиях, в частности, может быть взведен флажок ignoreLastWord и изменены поля лексемы, установленные акцептором
225. switch(lexem[0]){
226. case -1: //EOF
227. lexem[0]=0;break;
228. case -2: //операторыСравнения
230. break;
231. case -3: //знакОперации

234. break;
235. case -4: //ключевыеСлова
237. break;
238. case -5: //присваивание
240. break;
241. case -6: //логическиеОператоры
243. break;
244. case -7: //константа





251. break;
252. case -8: //ограничитель
254. break;
255. case -9: //скобки
257. break;
258. case -10: //словоДляФорматированияТекста
259. ignoreLastWord=true;
260. break;
261. case -11: //ввод/вывод

264. break;
265. case -12: //идентификатор
267. break;
268. case -13: //разделитель
270. break;
271. case -14: //комментарии
272. ignoreLastWord=true;
273. break;
274. }
275. }
276. lCnt += 1;
277. return lexem;
278. }
279. };
280. //end of Part\_4

283. var currentWord;
284. // синтаксический акцептор/анализатор
285. function Parser(reader){
286. this.currentLexem = [];
287. this.curWordIndex = 0;
288. this.la = new LexAnalyzer(reader);
289. this.cCnt = 0;
290. // заглушка для случая, когда нет синтаксических правил
291. };
292. Parser.prototype = {
293. getWordIndex: function(){
294. this.currentLexem = this.la.getLexem();
295. this.wList.push(this.currentLexem[1]);
296. return this.currentLexem[0];
297. },
298. getInfo: function(i){
299. switch (i) {
300. case 0:
301. return lCnt;
302. case 1:
303. return this.cCnt;
304. case 2:
305. return this.wList;
306. case 3:
307. return [];
308. }
309. return null;
310. },
311. parse: function(){
312. var wi;
313. this.wList = [];
314. lCnt = 0;
315. this.cCnt = 1;
316. while(((wi = this.getWordIndex()) != 0) && (wi != null) && (wi > -1000)){
317. if(this.cCnt++>998)
318. break;
319. }
320. return (wi == 0);
321. }
323. };
324. (function(){
325. document.getElementById("lexName").innerText = " автомат, управляемый таблицей состояний и переходов.";
326. document.getElementById("syntName").innerText = " заглушка.";
327. })();