МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники



**Лабораторная Работа №2**

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

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

Вариант № 31233114

|  |  |
| --- | --- |
| Выполнила: | Проверил: |
| Студентка гр. *АВТ-709*, *АВТФ* | *К.т.н., доцент* |
| *Черданцева Кристина* | *Малявко Александр Антонович* |
| «\_\_\_» \_\_\_\_\_\_ 2020г. | «\_\_\_» \_\_\_\_\_\_ 2020г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (подпись) | (подпись) |

Новосибирск

2020

**Оглавление**

[Цель работы 3](#_Toc35439667)

[1. Краткое описание языка регулярных выражений пакета ВебТрансЛаб 3](#_Toc35439668)

[2. Описание структуры программных модулей конечных автоматов, управляемых таблично и графом состояний и переходов 4](#_Toc35439669)

[3. Система регулярных выражений языка 6](#_Toc35439670)

[4. Описание языка 6](#_Toc35439671)

[5. Описание конечного автомата, реализованного с помощью графа состояний и переходов 8](#_Toc35439672)

[Выводы 8](#_Toc35439673)

[Приложение 8](#_Toc35439674)

# **Цель работы**

Цели 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**

**Первичное РВ**:

[<*произвольный символ*>] [a]; [a] → a

[<*перечень символов*>] [aA]; [aA] → а; [aA] → А .

[<*диапазон символов*>] [0–9]; [0–9] → 0 … [0–9] → 9

**Простое РВ:** это первичное РВ или произвольное РВ, заключенное в круглые скобки.

([a–zA–Z]) → a … ([a–zA–Z]) →z ([a–zA–Z]) → A … ([a–zA–Z]) → Z

Операции:

1. Конкатенация: [i][f] → if

2. Выбора: [0] | [1] ( эквивалентно [01] )

3. Ни одного или одно: РВ?

4. Одно или много: РВ+

5. Ни одного, одно или много: РВ\*

Идентификаторы: [*a–zA–Z*][*0–9a–zA–Z*]\* → indexOfSymbol

Вещественные константы: (([*0–9*]+ [.] [*0–9*]\*) | ([*0–9*]\* [.] [*0–9*]+)) → 3.14159

**Системы именованных регулярных выражений**

<имя группы слов> : <регулярное выражение>

Пример (RESystem1):

Ident: [a–zA–Z][0–9a–zA–Z]\*

Const: [0–9]+([.][0–9]\*)?

Const: [0–9]\*[.][0–9]+

Formatting: [ \r\n\t]+

Operation: [–+\*/]

Delimiter: [;]

Assign: [=]

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

**Конечный автомат, управляемый графом состояний и переходов**

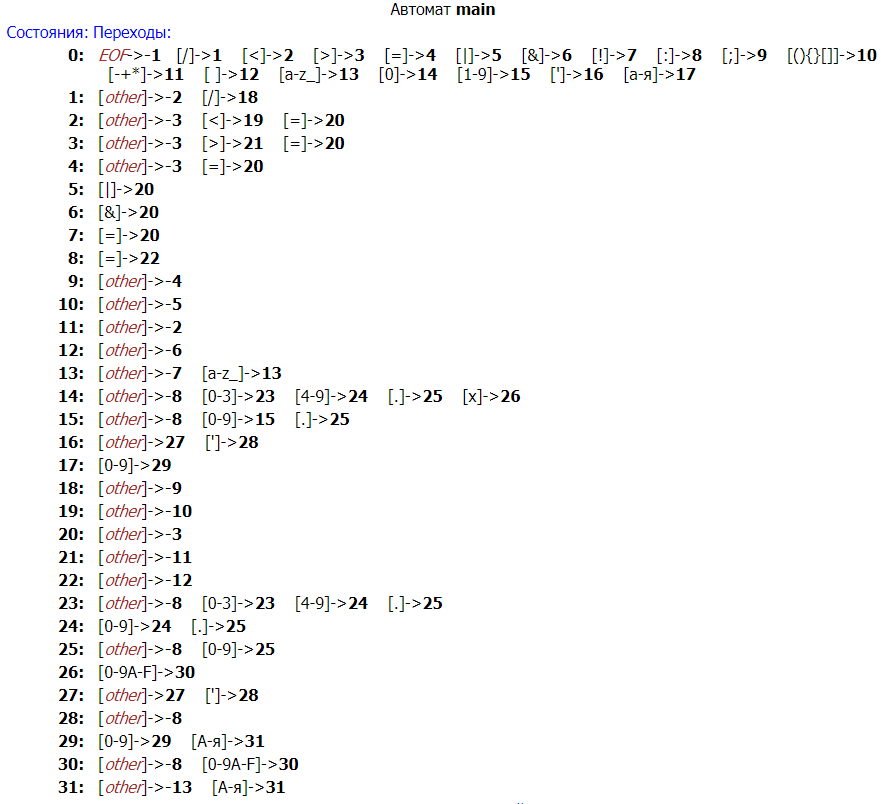


Рис. 1. Граф состояний и переходов лексического анализатора

Представляет собой список состояний и возможных переходов из них. На вход анализатора подается символ и обрабатывается в соответствии с графом состояний. На рисунке 2 указаны финальные состояния и действия. Как видно из рисунка 1, из нулевого состояния мы можем перейти в одно из состояний 1-17, и далее либо в рабочее, либо в финальное состояние.

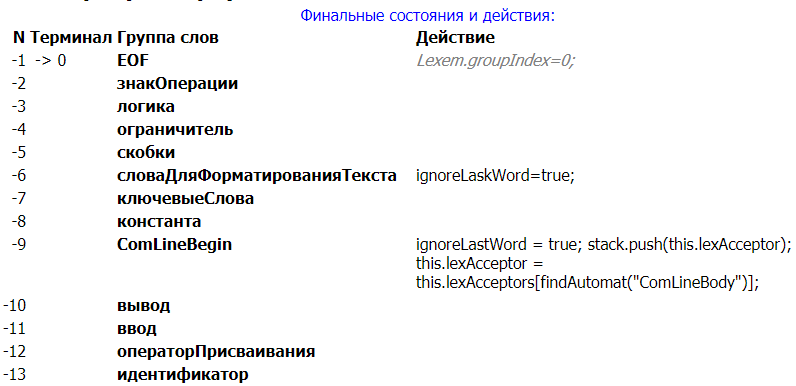


Рис. 2. Финальные состояния и действия

**Конечный автомат, управляемый таблично**

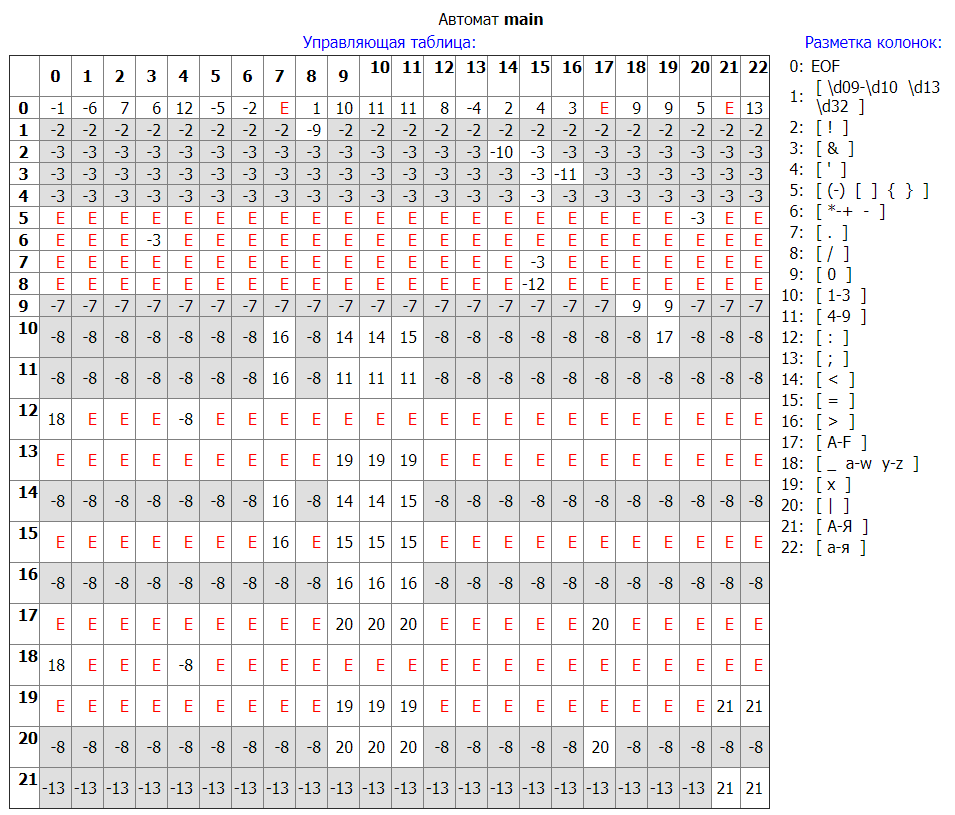


Рис. 3. Управляющая таблица

Представляет собой таблицу. Обозначения в клетках 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. Каждая строка таблицы соответствует своему состоянию. В столбцах таблицы видно, с каким результатом можно перейти в каждое из состояний. На рисунке 2 указаны финальные состояния и действия.

# **Система регулярных выражений языка**



Рис. 4. Правила лексики

# **Описание языка**

Программа на данном языке программирования располагается в одном файле. Программа состоит из функций, главной является функция main(). Выполнение программы начинается с функции main().

Синтаксис **функции** выглядит следующим образом: *тип\_данных имя\_функции* (*параметры*) {*тело функции* return *данные*;}. Тело функции начинается с открывающейся фигурной скобки и заканчивается закрывающейся фигурной скобкой.

Язык поддерживает только однострочные **комментарии** вида //*комментарий,* где // обозначает начало комментария, а ближайший последующий символ новой строки считается окончанием комментария.

**Оператор присваивания** в языке определяется как *имя\_переменной* := *значение*.

В языке присутствует **ограничитель** строки ;.

**Типы данных**

* Целый: int.
* Вещественный: float.
* Символьный: letter.
* Логический: bool, имеющий значения true или false.

Операции сравнения возвращают тип bool. Выражения в скобках после if, while приводятся к типу bool.

**Массивы**

Язык имеет структуру данных массив. Определяется он следующим образом: *тип\_данных* *идентификатор* [*размер*];.

**Константы**

Язык содержит целые, вещественные и строковые константы. Целые константы делятся на четверичные, десятичные и шестнадцатеричные. В виде регулярных выражений константы можно представить следующим образом: целые четверичные - [0][0-3]\*, целые десятичные - [1-9][0-9]\*, целые шестнадцатеричные - [0][x][0-9A-F]+, вещественные - [0-9]+[.][0-9]\*, строковые - ['][]\*['].

**Идентификаторы**

Заданный язык содержит идентификаторы, которые начинаются с одной маленькой русской буквы, далее следует непустая цепочка цифр и затем непустая цепочка русских букв любого регистра. В виде регулярного выражения можно представить так: [а-я][0-9]+[А-Яа-я]+.

**Арифметические операции**

В языке присутствуют такие арифметические операции, как умножение (\*), сложение (+), вычитание (-) и деление (/).

**Логические операции**

В языке присутствуют такие логические операции как сравнение (== или !=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=), а также логическое И (&&) и логическое ИЛИ (||).

**Условный оператор**

Условный оператор имеет вид:

when <ЛВ> then

<ОБ> [other <ОБ>]

<ЛВ> - логическое выражение, <ОБ> - оператор или блок. Блок заключается в фигурные скобки. Оператор other является необязательным и может отсутствовать.

**Оператор цикла**

Оператор цикла выглядит следующим образом:

loop <ОБ>

until <ЛВ>

<ЛВ> - логическое выражение, <ОБ> - оператор или блок.

**Оператор переключателя**

select <В> case (<K>)

<ОБ> [break;] …

[case () <ОБ>] end;

<ОБ> - оператор или блок, <K> - константа, <В> - выражение. Операторы в квадратных скобках являются необязательными.

**Ввод/вывод данных**

За ввод и вывод данных отвечают ключевые слова cin и cout. Синтаксис выглядит следующим образом: cin >> идентификатор; (считываем данные в переменную), cout << идентификатор; (выводим данные).

# **Описание конечного автомата, реализованного с помощью графа состояний и переходов**

Начинает работу из состояния 0. Читает символ из стека и определяет, в какое состояние перейти. Переходит в следующее состояние и считывает новый символ. Так до тех пор, пока не кончатся символы. Переход в финальное состояние является показателем окончания слова.

# **Выводы**

Описан заданный язык программирования. Доработана система регулярных выражений для всех групп слов языка. Построены программные модули, управляемые графом состояний и переходов и таблично. Изучена структура программных модулей. Подготовлены тексты программ на данном языке программирования.

# **Приложение**

***Приложение 1. Программа 1***

//минимум в массиве

int main() {

int н0н; // количество элементов в массиве

int а5а[н0н]; // массив целых чисел, максимальный размер 10 элементов

int и5и := 0;

int м1н; //минимум

cin >> н0н;

loop

cin >> а5а[и5и];

until (и5и < н0н)

м1н := а5а[0];

и5и := 0;

loop

when (а5а[и5и] < м1н) then м1н := а5а[и5и];

и5и := и5и + 1;

until (и5и < н0н)

cout << м1н;

return 0;

}

***Приложение 2. Программа 2***

//перевод из 16 в 4 сс

int main(){

int а16а, а4а := 0;

cin >> а16а;

select а16а

case (0xA)

а4а := 022;

break;

case (0xB)

а4а := 023;

break;

case (0xC)

а4а := 030;

break;

case (0xD)

а4а := 031;

break;

case (0xE)

а4а := 032;

break;

case (0xF)

а4а := 033;

break;

end;

cout << а4а;

return 0;

}

***Приложение 3. Текст автомата, реализованного таблично***

//Построено 18.3.2020 11:13:06 по исходному файлу/шаблону lr1.xml/lexAsTableSyntAsSingleFSM\_to\_js

var lCnt = 0;

var cCnt = 0;

//входной поток литер

var TextReader = {

text: "^",

len: -1,

backedChar: 0,

pos: 0,

setText: function(t){

this.text = t;

this.len = t.length - 1;

this.pos = 0;

},

//возврат заданной литеры во входной поток

back: function(bc){

if(this.pos > 0){ //если нет возвращенной, но не прочитанной литеры

this.pos = -this.pos; //устанавливаем признак наличия возвращенной литеры

this.backedChar = bc; //и запоминаем ее код

}

},

//чтение одной литеры

read: function() {

if(this.pos >= 0){ //если нет возвращенной литеры

//и возвращаем -1, если достигнут конец входного потока (или входного потока просто нет), или очередную литеру из него

return this.pos <= this.len? this.text[this.pos++]:null;

}

else //если есть возвращенная литера

{

this.pos = -this.pos; //сбросим признак ее наличия

return this.backedChar; //и вернем ее вновь

}

}

};

// лексический акцептор

/\*\* Преобразователь кода символа в индекс.

\* Использует собственный вектор диапазонов

\* (первый символ, последний символ, индекс, счетчик преобразований)

\* Диапазоны могут добавляться и удаляться на лету.

\* При преобразовании диапазоны могут переупорядочиваться согласно частоте появления символов.

\* В результате переупорядочивания диапазоны, содержащие часто появляющиеся символы,

\* оказываются в начале вектора, а те, символы которых появляются редко, вытесняются в его конец

\*/

function CharToIndex() {

this.count = 0;

this.fChar = [];

this.lChar = [];

this.index = [];

// charCnt = new int[limit];

// byChar = new int[limit];

};

/\*\*Создается новый диапазон символов.

\* @param fChr первый символ диапазона

\* @param lChr последний символ диапазона

\* @param indexChr индекс, приписываемый диапазону

\*/

CharToIndex.prototype = {

addRange: function(fChr, lChr, indexChr){

//запись параметров диапазона в первое свободное место

this.fChar[this.count] = fChr;

this.lChar[this.count] = lChr;

this.index[this.count++] = indexChr;

},

getIndexOfChar: function(Chr) {

var i;

for(i = 0; i < this.count; i++)

if((Chr >= this.fChar[i]) && (Chr <= this.lChar[i])) //символ больше/равен началу диапазона?

return this.index[i]; //вернем индекс

return -1; //возращается, ели не найден диапазон.

}

};

//конструктор

function fAutomatAsTable(no, cToI, ctlTbl, tr){

this.index = no; //запомним собственный индекс

this.charToI = cToI;

this.controlTable = ctlTbl;

this.inp = tr;

this.w = "";//мультиавтоматный вариант

this.hasOther = true;

}

fAutomatAsTable.prototype = {

//мультиавтоматный вариант

setOther(flag){

this.hasOther = flag;

},

//основной метод лексического акцептора - чтение одного слова из входного потока и формирование лексемы

getLexem: function(){

var curState = 0;

var newState = 0;

var curChar = -1;

var colIndex; //объявление и инициализация локальных переменных

this.w = "";

for( ; ; ) { //запускаем цикл имитации работы конечного автомата, выполняющийся до тех пор, пока не было перехода в финальное состояние или не обнаружена ошибка

if((curChar = this.inp.read()) === null) //читаем очередную входную литеру; если возвращен признак исчерпания входного потока (-1)

colIndex = 0; //то устанавливаем колонку УТ, соответствующую эпсилон

else

colIndex = this.charToI.getIndexOfChar(curChar); //преобразуем код литеры в индекс колонки УТ

if((colIndex < 0) || (colIndex >= this.controlTable[curState].length)) //если индекс колонки находится за пределами УТ

//мультиавтоматный вариант

if(this.hasOther)

colIndex = this.controlTable[curState].length - 1;//устанавливаем его на последнюю колонку, формируемую специально для реализации выражений other[+] в мультиавтоматном варианте

else {

curState = -2147483647;

break;

}

newState = this.controlTable[curState][colIndex]; //выбираем из УТ номер следующего состояния

if((newState == -2147483647) && (this.controlTable[curState][0] > 0))

newState = this.controlTable[curState][0];

curState = newState;

if((curChar === null) || (curState < 0)) //если исчерпан входной поток или автомат перходит в финальное состояние или обнаружена ошибка

break; //выходим из цикла

this.w += curChar; //добавляем литеру к тексту слова

}

if((curState & 0x40000000) != 0)

this.inp.back(curChar);

else

this.w += curChar; //добавляем литеру к тексту слова

return [(curState >= 0 ? -2147483647 : curState | 0x40000000), this.w]; //вернем лексему

},

getIndex: function() {

return this.index;

}

};

// лексический анализатор

// данные из правил

var stack = [];

function LexAnalyzer(rdr) {

this.controlTable0 = [];

this.charToI0 = new CharToIndex();

this.controlTable1 = [];

this.charToI1 = new CharToIndex();

this.lexStk = [];

this.lexAcceptors = new Array(2);

//далее - формирование всех управляющих таблиц

//УТ автомата main

this.controlTable0.push([-1,-1073741830,7,6,12,-1073741829,-1073741826,-2147483647,1,10,11,11,8,-1073741828,2,4,3,-2147483647,9,9,5,-2147483647,13,0]);

this.controlTable0.push([-2,-2,-2,-2,-2,-2,-2,-2,-1073741833,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,0]);

this.controlTable0.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741834,-1073741827,-3,-3,-3,-3,-3,-3,-3,0]);

this.controlTable0.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-1073741835,-3,-3,-3,-3,-3,-3,0]);

this.controlTable0.push([-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-1073741827,-3,-3,-3,-3,-3,-3,-3,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741827,-2147483647,-2147483647,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-1073741827,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741827,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-1073741836,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,-7,9,9,-7,-7,-7,0]);

this.controlTable0.push([-8,-8,-8,-8,-8,-8,-8,16,-8,14,14,15,-8,-8,-8,-8,-8,-8,-8,17,-8,-8,-8,0]);

this.controlTable0.push([-8,-8,-8,-8,-8,-8,-8,16,-8,11,11,11,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,0]);

this.controlTable0.push([18,-2147483647,-2147483647,-2147483647,-1073741832,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,19,19,19,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-8,-8,-8,-8,-8,-8,-8,16,-8,14,14,15,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,16,-2147483647,15,15,15,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-8,-8,-8,-8,-8,-8,-8,-8,-8,16,16,16,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,-8,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,20,20,20,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,20,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([18,-2147483647,-2147483647,-2147483647,-1073741832,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,0]);

this.controlTable0.push([-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,19,19,19,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,-2147483647,21,21,0]);

this.controlTable0.push([-8,-8,-8,-8,-8,-8,-8,-8,-8,20,20,20,-8,-8,-8,-8,-8,20,-8,-8,-8,-8,-8,0]);

this.controlTable0.push([-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,-13,21,21,0]);

//УТ автомата ComLineBody

this.controlTable1.push([-1,-1073741827,1,0]);

this.controlTable1.push([-2,-2,1,0]);

//здесь - формирование транслитераторов

//транслитератор автомата main

this.charToI0.addRange('\u0009','\n',1);

this.charToI0.addRange('\r','\r',1);

this.charToI0.addRange(' ',' ',1);

this.charToI0.addRange('!','!',2);

this.charToI0.addRange('&','&',3);

this.charToI0.addRange('\'','\'',4);

this.charToI0.addRange('(',')',5);

this.charToI0.addRange('[','[',5);

this.charToI0.addRange(']',']',5);

this.charToI0.addRange('{','{',5);

this.charToI0.addRange('}','}',5);

this.charToI0.addRange('\*','+',6);

this.charToI0.addRange('-','-',6);

this.charToI0.addRange('.','.',7);

this.charToI0.addRange('/','/',8);

this.charToI0.addRange('0','0',9);

this.charToI0.addRange('1','3',10);

this.charToI0.addRange('4','9',11);

this.charToI0.addRange(':',':',12);

this.charToI0.addRange(';',';',13);

this.charToI0.addRange('<','<',14);

this.charToI0.addRange('=','=',15);

this.charToI0.addRange('>','>',16);

this.charToI0.addRange('A','F',17);

this.charToI0.addRange('\_','\_',18);

this.charToI0.addRange('a','w',18);

this.charToI0.addRange('y','z',18);

this.charToI0.addRange('x','x',19);

this.charToI0.addRange('|','|',20);

this.charToI0.addRange('А','Я',21);

this.charToI0.addRange('а','я',22);

//транслитератор автомата ComLineBody

this.charToI1.addRange('\n','\n',1);

this.charToI1.addRange('\r','\r',1);

//и, наконец, - создание всех лексических акцепторов

//конечный автомат main

this.lexAcceptors[0] = new fAutomatAsTable(0,this.charToI0,this.controlTable0,rdr);//мультиавтоматный вариант

this.lexAcceptors[0].setOther(0);

//конечный автомат ComLineBody

this.lexAcceptors[1] = new fAutomatAsTable(1,this.charToI1,this.controlTable1,rdr);//мультиавтоматный вариант

this.lexAcceptors[1].setOther(1);

this.lexAcceptor = this.lexAcceptors[0]; //текущим установим автомат с индексом 0

this.lexStk.push(this.lexAcceptor); //и запомним его в стеке

}

// распознавание слова и формирование лексемы

LexAnalyzer.prototype = {

getLexem: function(){//мультиавтоматный вариант

var groupIndex;

var ignoreLastWord;

var lexem;

if(this.lexStk.length<=0){ //если стек пуст - вернем лексему-ошибку

groupIndex = -2147483647;

lCnt += 1;

return [groupIndex, null];

}

ignoreLastWord = true; //для входа в основной цикл установим флажок игнорирования слова

while(ignoreLastWord) {

ignoreLastWord = false; //и сбросим его (возможно, он будет установлен действием)

//вызов лексического акцептора:

lexem = this.lexAcceptor.getLexem(); //получим лексему от текущего лексического акцептора

//для каждого автомата и каждой группы слов обеспечим возможность выполнения действий (имена автоматов и групп слов в примечаниях)

//индексы групп слов, имена которых используются в грамматике, преобразуются в те значения, который присвоил им построитель синтаксического акцептора

//в действиях, в частности, может быть взведен флажок ignoreLastWord и изменены поля лексемы, установленные акцептором

switch(this.lexAcceptor.getIndex()) {

case 0: // Автомат: main

switch(lexem[0]) {

case -1: //EOF

lexem[0] = 0;break;

case -2: //знакОперации

break;

case -3: //логика

break;

case -4: //ограничитель

break;

case -5: //скобки

break;

case -6: //словаДляФорматированияТекста

ignoreLaskWord=true;

break;

case -7: //ключевыеСлова

break;

case -8: //константа

break;

case -9: //ComLineBegin

ignoreLastWord = true;

stack.push(this.lexAcceptor);

this.lexAcceptor = this.lexAcceptors[1];

break;

case -10: //вывод

break;

case -11: //ввод

break;

case -12: //операторПрисваивания

break;

case -13: //идентификатор

break;

}

break;

case 1: // Автомат: ComLineBody

switch(lexem[0]) {

case -1: //EOF

lexem[0] = 0;break;

case -2: //other

ignoreLastWord = true;

break;

case -3: //ComLineEnd

ignoreLastWord = true;

this.lexAcceptor = stack.pop();

break;

}

break;

}

}

lCnt += 1;

return lexem;

}

};

var currentWord;

// синтаксический акцептор/анализатор

function Parser(rdr){

this.la = new LexAnalyzer(rdr);

this.currentLexem = []; //текущая лексема

this.wList = [];

this.cCnt=0;

// заглушка для случая, когда нет синтаксических правил

};

Parser.prototype = {

getWordIndex: function(){

this.currentLexem = this.la.getLexem();

this.wList.push(this.currentLexem[1]);

return this.currentLexem[0];

},

getInfo: function(i){

switch (i) {

case 0:

return lCnt;

case 1:

return this.cCnt;

case 2:

return this.wList;

case 3:

return [];

}

return null;

},

parse: function(){

var wi;

this.wList = [];

this.cCnt = 1;

lCnt = 0;

while(((wi = this.getWordIndex()) != 0) && (wi != null) && (wi > -100)){

if(this.cCnt++>998)

break;

}

return (wi == 0);

}

};

(function(){

document.getElementById("lexName").innerText = " автомат, управляемый таблицей состояний и переходов.";

document.getElementById("syntName").innerText = " Заглушка.";

})();

***Приложение 4. Текст автомата, графовая реализация***

//Построено 18.3.2020 12:50:47 по исходному файлу/шаблону lr1.xml/lexAsGraphSyntAsSxLR\_to\_js

var ERROR\_VALUE = false;

var lCnt = 0;

var cCnt = 0;

var stk = [];

//входной поток литер

var TextReader = {

text: "^",

len: -1,

backedChar: 0,

pos: 0,

setText: function(t){

this.text = t;

this.len = t.length - 1;

this.pos = 0;

},

//возврат заданной литеры во входной поток

back: function(bc){

if(this.pos > 0){ //если нет возвращенной, но не прочитанной литеры

this.pos = -this.pos; //устанавливаем признак наличия возвращенной литеры

this.backedChar = bc; //и запоминаем ее код

}

},

//чтение одной литеры

read: function() {

if(this.pos >= 0){ //если нет возвращенной литеры

//и возвращаем -1, если достигнут конец входного потока (или входного потока просто нет), или очередную литеру из него

return this.pos <= this.len? this.text[this.pos++]:null;

}

else //если есть возвращенная литера

{

this.pos = -this.pos; //сбросим признак ее наличия

return this.backedChar; //и вернем ее вновь

}

}

};

// лексический акцептор

//вспомогательные классы

//вершина графа состояний и переходов - состояние конечного автомата

function FAstate(toCount){

this.count = toCount;

this.markTo = []; //создание массива дуг

this.statesTo = []; //создание массива номеров вершин

this.filled=0; //установка нулевой степени заполненности массивов

}

FAstate.prototype = {

//добавление дуги, ведущей в состояние to с разметкой mark

setArc: function(to, mark){

if(this.filled < this.count){ //если в массивах есть место для новой дуги

this.statesTo[this.filled] = to; //запомним, куда она ведет

this.markTo[this.filled++] = mark;

}

},

//определить, ведет ли в какое-нибудь состояние дуга по заданной литере

getState: function(c) {

var i;

var forAll = 0;

for(i = 0; i < this.filled; i++){ //просмотреть все дуги

if(this.markTo[i] == "")

forAll = this.statesTo[i];

if(this.markTo[i].indexOf(c) >= 0)

return this.statesTo[i];

}

return forAll == 0? null:forAll;

},

//вернуть номер финального состояния, если в такое состояние ведет хотя бы одна дуга

getFinalState: function(){

var i;

for(i = 0; i < this.filled; i++){ //просмотреть все дуги

if(this.statesTo[i] < 0) //если дуга ведет в финальное состояние

return this.statesTo[i]; // вернуть его номер

}

return null; //если нет дуг в финальные состояния - вернуть максимально возможное отрицательное число

}

};

//лексический акцептор, управляемый графом состояний и переходов

//конструктор

function Fautomat(no, ind, inp){

this.index = no;

this.reader = inp;

this.states = [];

}

Fautomat.prototype = {

//сохранение заданного состояния (вершины)

setState: function(no, state){

this.states[no] = state; //сохраним состояние в массиве

},

//основной метод лексического акцептора - чтение одного слова из входного потока и формирование лексемы

getLexem: function(){

var curState; //текущее состояние (-1 на случай, если входной поток ранее был исчерпан)

var curChar; //текущая литеа (-1 на случай, если входной поток ранее был исчерпан)

var newState;

var word = "";

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

curChar = this.reader.read(); //пытаемся получить очередную литеру

if(curChar === null) { //если нет больше литер в тексте

//формируем текущее состояние

//-1, если конец входного потока обнаружен в момент запуска автомата

//и номер финального состояния, в которое из текущего есть переход по эпсилон в противном случае

//(возможен случай, когда такого перехода нет, тогда получим значение -2147483647)

curState = (curState == 0 ? -1 : this.states[curState].getFinalState());

break;

} else { //если код литеры получен

newState = this.states[curState].getState(curChar); //получаем новый номер состояния

if(newState === null)

curState = this.states[curState].getFinalState();

else

curState = newState;

//если номер нового состояния вне пределов массива состояний или код литеры меньше нуля - выйдем из цикла

if((curState < 0) || (curState === null) || (curState >= this.states.length))

break;

word += curChar; //если по этой литере был выполнен переход в рабочее состояние - добавим ее к тексту слова

}

}

if(curState < -1) //если не EndOfFile и не ошибка (в частности ошибкой является случай, когда curState>=0)

this.reader.back(curChar); //вернем литеру во входной поток

return [curState, word]; //вернем лексему

},

//возвращает индекс акцептора, нужен в мультиавтоматном варианте

getIndex: function() { //возвращает индекс акцептора, нужен в мультиавтоматном варианте

return this.index;

}

};

// Лексический анализатор

var stack = [];

function LexAnalyzer(rdr) {

var state;

this.lexAcceptors = [];

//дальше идет создание автоматов, их состояний, выходящих из них дуг и запоминание всего этого в соответствующих местах

this.lexAcceptors[0] = new Fautomat(0,32,rdr);

state = new FAstate(17);state.setArc(1, "/");state.setArc(2, "<");state.setArc(3, ">");state.setArc(4, "=");state.setArc(5, "|");state.setArc(6, "&");state.setArc(7, "!");state.setArc(8, ":");state.setArc(9, ";");state.setArc(10, "(){}[]");state.setArc(11, "-+\*");state.setArc(12, " \n\r\u0009");state.setArc(13, "abcdefghijklmnopqrstuvwxyz\_");state.setArc(14, "0");state.setArc(15, "123456789");state.setArc(16, "\'");state.setArc(17, "абвгдежзийклмнопрстуфхцчшщъыьэюя");

this.lexAcceptors[0].setState(0,state);

state = new FAstate(2);state.setArc(-2, "");state.setArc(18, "/");

this.lexAcceptors[0].setState(1,state);

state = new FAstate(3);state.setArc(-3, "");state.setArc(19, "<");state.setArc(20, "=");

this.lexAcceptors[0].setState(2,state);

state = new FAstate(3);state.setArc(-3, "");state.setArc(21, ">");state.setArc(20, "=");

this.lexAcceptors[0].setState(3,state);

state = new FAstate(2);state.setArc(-3, "");state.setArc(20, "=");

this.lexAcceptors[0].setState(4,state);

state = new FAstate(1);state.setArc(20, "|");

this.lexAcceptors[0].setState(5,state);

state = new FAstate(1);state.setArc(20, "&");

this.lexAcceptors[0].setState(6,state);

state = new FAstate(1);state.setArc(20, "=");

this.lexAcceptors[0].setState(7,state);

state = new FAstate(1);state.setArc(22, "=");

this.lexAcceptors[0].setState(8,state);

state = new FAstate(1);state.setArc(-4, "");

this.lexAcceptors[0].setState(9,state);

state = new FAstate(1);state.setArc(-5, "");

this.lexAcceptors[0].setState(10,state);

state = new FAstate(1);state.setArc(-2, "");

this.lexAcceptors[0].setState(11,state);

state = new FAstate(1);state.setArc(-6, "");

this.lexAcceptors[0].setState(12,state);

state = new FAstate(2);state.setArc(-7, "");state.setArc(13, "abcdefghijklmnopqrstuvwxyz\_");

this.lexAcceptors[0].setState(13,state);

state = new FAstate(5);state.setArc(-8, "");state.setArc(23, "0123");state.setArc(24, "456789");state.setArc(25, ".");state.setArc(26, "x");

this.lexAcceptors[0].setState(14,state);

state = new FAstate(3);state.setArc(-8, "");state.setArc(15, "0123456789");state.setArc(25, ".");

this.lexAcceptors[0].setState(15,state);

state = new FAstate(2);state.setArc(27, "");state.setArc(28, "\'");

this.lexAcceptors[0].setState(16,state);

state = new FAstate(1);state.setArc(29, "0123456789");

this.lexAcceptors[0].setState(17,state);

state = new FAstate(1);state.setArc(-9, "");

this.lexAcceptors[0].setState(18,state);

state = new FAstate(1);state.setArc(-10, "");

this.lexAcceptors[0].setState(19,state);

state = new FAstate(1);state.setArc(-3, "");

this.lexAcceptors[0].setState(20,state);

state = new FAstate(1);state.setArc(-11, "");

this.lexAcceptors[0].setState(21,state);

state = new FAstate(1);state.setArc(-12, "");

this.lexAcceptors[0].setState(22,state);

state = new FAstate(4);state.setArc(-8, "");state.setArc(23, "0123");state.setArc(24, "456789");state.setArc(25, ".");

this.lexAcceptors[0].setState(23,state);

state = new FAstate(2);state.setArc(24, "0123456789");state.setArc(25, ".");

this.lexAcceptors[0].setState(24,state);

state = new FAstate(2);state.setArc(-8, "");state.setArc(25, "0123456789");

this.lexAcceptors[0].setState(25,state);

state = new FAstate(1);state.setArc(30, "0123456789ABCDEF");

this.lexAcceptors[0].setState(26,state);

state = new FAstate(2);state.setArc(27, "");state.setArc(28, "\'");

this.lexAcceptors[0].setState(27,state);

state = new FAstate(1);state.setArc(-8, "");

this.lexAcceptors[0].setState(28,state);

state = new FAstate(2);state.setArc(29, "0123456789");state.setArc(31, "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя");

this.lexAcceptors[0].setState(29,state);

state = new FAstate(2);state.setArc(-8, "");state.setArc(30, "0123456789ABCDEF");

this.lexAcceptors[0].setState(30,state);

state = new FAstate(2);state.setArc(-13, "");state.setArc(31, "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя");

this.lexAcceptors[0].setState(31,state); this.lexAcceptors[1] = new Fautomat(1,3,rdr);

state = new FAstate(2);state.setArc(1, "");state.setArc(2, "\n\r");

this.lexAcceptors[1].setState(0,state);

state = new FAstate(2);state.setArc(-2, "\n\r");state.setArc(1, "");

this.lexAcceptors[1].setState(1,state);

state = new FAstate(1);state.setArc(-3, "");

this.lexAcceptors[1].setState(2,state);

this.lexAcceptor = this.lexAcceptors[0]; //установим автомат с индексом 0 в качестве стартового лексического акцептора

this.lexStk = [this.lexAcceptor]; //и запомним это в стеке

};

// распознавание слова и формирование лексемы

LexAnalyzer.prototype = {

getAcceptor: function(){

return this.lexAcceptor;

},

getLexem: function() {

if(this.lexStk.length <= 0){ //если стек пуст - вернуть лексему с ошибкой

lCnt += 1;

return [null,null];

}

var ignoreLastWord = true; //временно установим флажок, чтобы войти в цикл

var lexem;

while(ignoreLastWord){ //пока флажок установлен - читаем очередное слово

ignoreLastWord = false; //перед чтением следующего слова сбросим флажок

//вызовем лексический акцептор и получим от него лексему:

lexem = this.lexAcceptor.getLexem();

//для каждого автомата и каждой группы слов обеспечим возможность выполнения действий (имена автоматов и групп слов в примечаниях)

//индексы групп слов, имена которых используются в грамматике, преобразуются в те значения, который присвоил им построитель синтаксического акцептора

//в действиях, в частности, может быть взведен флажок ignoreLastWord и изменены поля лексемы, установленные акцептором

switch(this.lexAcceptor.getIndex()){

case 0: // Автомат: main

switch(lexem[0]) {

case -1: //EOF

lexem[0] = 0;

break;

case -2: //знакОперации

break;

case -3: //логика

break;

case -4: //ограничитель

break;

case -5: //скобки

break;

case -6: //словаДляФорматированияТекста

ignoreLaskWord=true;break;

case -7: //ключевыеСлова

break;

case -8: //константа

break;

case -9: //ComLineBegin

ignoreLastWord = true;

stack.push(this.lexAcceptor);

this.lexAcceptor = this.lexAcceptors[1];break;

case -10: //вывод

break;

case -11: //ввод

break;

case -12: //операторПрисваивания

break;

case -13: //идентификатор

break;

}

break;

case 1: // Автомат: ComLineBody

switch(lexem[0]) {

case -1: //EOF

lexem[0] = 0;

break;

case -2: //other

ignoreLastWord = true;break;

case -3: //ComLineEnd

ignoreLastWord = true;

this.lexAcceptor = stack.pop();break;

}

break;

}

}

lCnt += 1;

return lexem;

}

};

var la;

var words;

var wList;

var stk = [];

var currentLexem = [];

var currentWord;

var curWordIndex;

var stackPush = function(val){

stk.push(val);

};

var stackPop = function(cnt){

for(;cnt>0;cnt-=1)

if(stk.length > 0)

stk.pop();

else

return true;

return false;

};

var stateFromStack = function(){

return ((stk.length <= 0)? -1 :stk[stk.length-1]);

};

// синтаксический акцептор/анализатор

function Parser(rdr){

stk = [];

currentLexem = [];

la = new LexAnalyzer(rdr); //экземпляр лексического анализатора

wList = [];

this.pHist = [];

this.cCnt = 0; //начальное значение счетчика циклов

// заглушка для случая, когда нет синтаксических правил

};

Parser.prototype = {

getWordIndex: function(){

currentLexem = la.getLexem();

this.wList.push(currentLexem[1]);

return currentLexem[0];

},

getInfo: function(i){

switch (i) {

case 0:

return lCnt;

case 1:

return this.cCnt;

case 2:

return this.wList;

case 3:

return [];

}

return null;

},

parse: function(){

var wi;

lCnt = 0;

this.cCnt = 1;

this.wList = [];

while(((wi = this.getWordIndex()) != 0) && (wi != null) && (wi > -1000)){

if(this.cCnt++>998)

break;

}

return (wi == 0);

}

};

(function(){

document.getElementById("lexName").innerText = " автомат, управляемый графом состояний и переходов.";

document.getElementById("syntName").innerText = " заглушка.";

})();