МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники



**Лабораторная Работа №4**

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

**на тему:** *«Синтаксис языков программирования. Нисходящий синтаксический анализ»*

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](#_Toc37701305)

[Реализация LL(1)-грамматики 3](#_Toc37701306)

[Описание грамматики 4](#_Toc37701307)

[Текст процедурной реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора, построенных по этой грамматике, описание алгоритмов работы соответствующих автоматов 5](#_Toc37701308)

[Фрагменты историй работы процедурной и автоматных реализаций нисходящего синтаксического акцептора для правильного и ошибочного тестовых примеров с объяснением принципов работы каждого акцептора 7](#_Toc37701309)

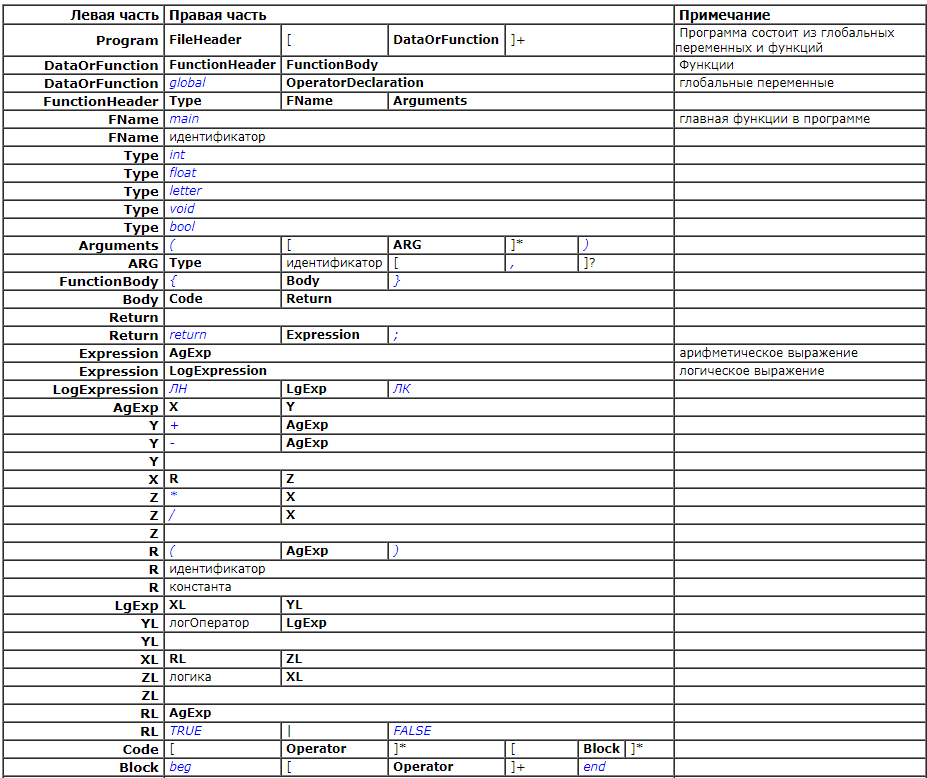
[Выводы 13](#_Toc37701310)

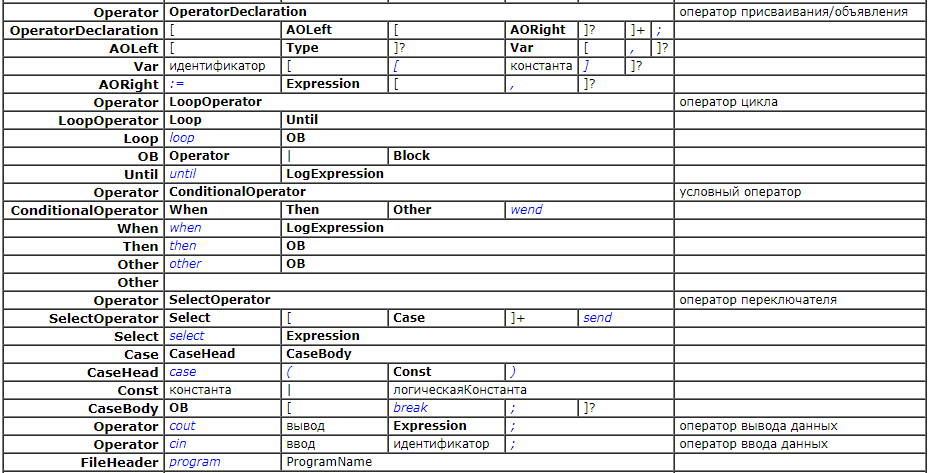
[Приложение 14](#_Toc37701311)

# **Цель работы**

Изучение основных идей и понятий нисходящих методов синтаксического анализа, выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобретение навыков построения процедурной и различных автоматных реализаций нисходящего анализа, исследование поведения нисходящих синтаксических акцепторов.

# **Реализация LL(1)-грамматики**





# **Описание грамматики**

Программа состоит из заголовка и данных или функций. Заголовок программы представляет из себя ключевое слово Program и название программы. В качестве данных используются глобальные переменные: global оператор объявления переменных.

В программе может быть одна или несколько функций, главной из которых является функция main. Функция имеет заголовок и тело. Заголовок функции состоит из типа функции, имени функции и аргументов. Существует 5 типов данных: int, float, letter, void, bool. Название функции может быть либо main, либо идентификатор. Аргументы функции могут отсутствовать, либо есть один или несколько аргументов. Они перечисляются в круглых скобках после имени функции, выглядят следующим образом: тип данных идентификатор.

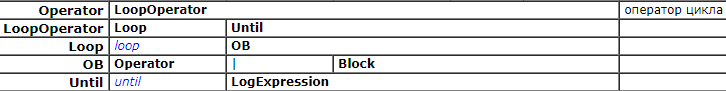
Тело функции может содержать какой-то код и возвращать какое-либо значение или выражение. Код в теле функции состоит из операторов и блоков. К операторам относятся: оператор объявления/присваивания, оператор цикла, условный оператор, оператор переключателя, операторы ввода и вывода данных. Блок состоит из одного или нескольких операторов, заключенных между ключевых слов beg и end.

Оператор объявления/присваивания выглядит следующим образом:



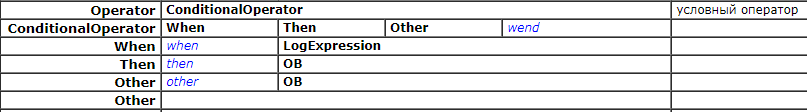
В нем обязательно присутствует идентификатор и точка с запятой. Можно объявить несколько переменных в одном операторе и сразу присвоить им значение.

Оператор цикла выглядит следующим образом:



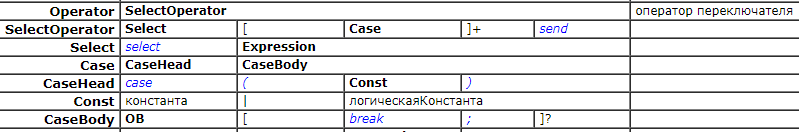
Начинается с ключевого слова loop, далее идет оператор или блок, а затем условие окончания цикла until лигоческое\_выражение.

Условный оператор выглядит следующим образом:



Обязательными частями являются When и Then, Other может отсутствовать. В конце оператора обязательно присутствует ключевое слово wend.

Оператор переключателя выглядит следующим образом:



Обязательной частью является Select, Case должно присутствовать хотя бы один раз, в конце оператора должно присутствовать ключевое слово send.

Операторы ввода и вывода выглядят следующим образом:



В программе могут использоваться арифметические и логические выражения. Логические выражения заключаются между двух слов ЛН и ЛК.

# **Текст процедурной реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора, построенных по этой грамматике, описание алгоритмов работы соответствующих автоматов**

*Конечный автомат со стековой памятью и несколькими состояниями*

Управляющая таблица в приложении 1.

Функционирование конечного автомата со стековой памятью и несколькими состояниями определяется управляющей таблицей. Предполагается, что автомат при запуске оказывается в особом начальном состоянии, на каждом такте по входному символу и текущему состоянию определяет и выполняет операции над входным потоком символов, стековой памятью и собственным состоянием.

Каждому символу правила грамматики должно быть поставлено в соответствие в точности одно состояние автомата. С каждым состоянием должно быть связано множество выбора и два адреса перехода (один используется при положительном результате проверки принадлежности текущего символа множеству выбора, второй – при отрицательном). Под адресом перехода понимается номер состояния.

С каждым состоянием должны быть также связаны операции управления стековой памятью (задание адреса возврата, снятие адреса с верхушки стека и переключение в состояние возврата) и операция управления чтением следующего входного символа. Все операции управления могут задаваться булевскими значениями true/false (флажками). Обозначения для флажков управления операциями:

* Флажок **а** управляет чтением следующего входного символа;
* Флажок **s** управляет занесением адреса точки возврата (вычисляемого как номер текущего состояния + 1) в стек;
* Флажок **r** обеспечивает переключение автомата в состояние, номер которого снимается с верхушки стека возвратов;
* Флажок **е** запрещает останов по ошибке в случае, когда состояние соответствует нетерминалу из левой части и есть еще хотя бы одно правило для такого нетерминала.

Каждая клетка управляющей таблицы автомата должна содержать следующие поля:



*Конечный автомат со стековой памятью и одним состоянием*

(Управляющая таблица слишком большая, не получилось вставить ее в отчет)

Поведение автомата определяется управляющей таблицей, столбцы которой соответствуют входным символам, строки – символам, которые могу находиться в стеке, а в клетках указана некоторая последовательность операция над стеком, входным потоком и состоянием автомата.

Операции для стековой памяти:

* Занесение символа в стек;
* Снятие одного символа с верхушки стека.

Над входным потоком определена единственная операция: чтение следующего символа из входной цепочки.

Для управления состоянием будем использовать единственный знак операции Stop, предназначенный для останова автомата по успешному окончанию восстановления дерева. Для обозначения операции останова по обнаружению ошибок во входной цепочке будем использовать обычное соглашение: клетка управляющей таблицы пуста.

*Процедурная реализация рекурсивного спуска*

Текст процедурной реализации в приложении 2.

Процедурная реализация рекурсивного спуска – LL1-грамматика, преобразованная в программу синтаксического акцептора.

Рекурсивный спуск может быть реализован только на языке программирования, который допускает рекурсивный вызов функций.

Имеется глобальная переменная, предназначенная для хранения очередного терминального символа входной цепочки. Также имеется функция чтения очередного символа из этой цепочки, по существу, лексический анализатор.

Рекурсивный синтаксический акцептор – это функция, возвращающая логическое значение (true – если входное предложение правильное, false – в противном случае).

Каждому нетерминалу грамматики ставится в соответствие функция, выполняющая действия по проверке правильности терминальных цепочек, выводимых из этого нетерминала, определяемые совокупностью правил для него и текущим терминалом.

При процедурной реализации нисходящего синтаксического акцепта уровни восстановления дерева разбора как цепочки символов в явном виде нигде не хранятся. Неявно эти цепочки для каждого шага работы акцептора полностью определены текущим содержанием стека возвратов. В нем хранятся точки возврата, из которых должно продолжаться выполнение функций акцептора, с каждой из таких точек можно сопоставить ожидаемые терминальные или нетерминальные символы.

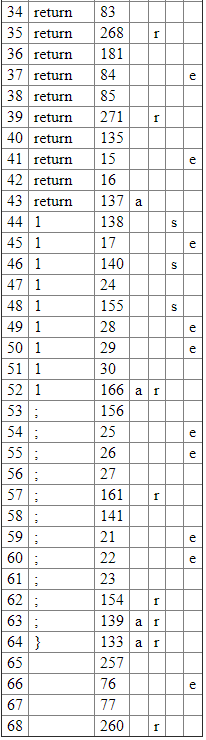
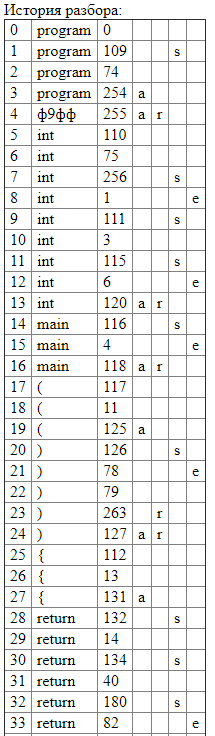
# **Фрагменты историй работы процедурной и автоматных реализаций нисходящего синтаксического акцептора для правильного и ошибочного тестовых примеров с объяснением принципов работы каждого акцептора**

*Пример правильной программы:*

program ф9фф

int main() { return 1;}

Конечный автомат со стековой памятью и несколькими состояниями:

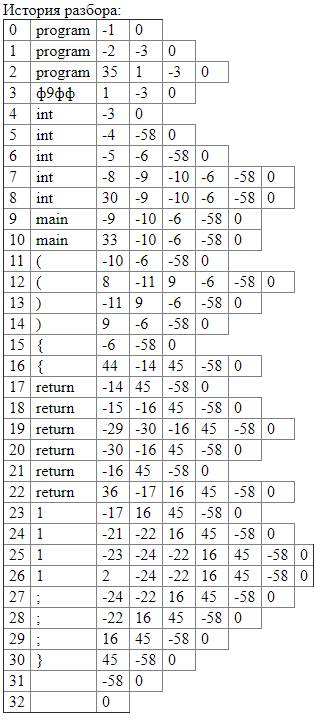


В первом столбце находится номер такта, во втором столбце таблицы находится множество выбора состояний, в третьем – адрес перехода, 4-7 – флажки.

Алгоритм работы:

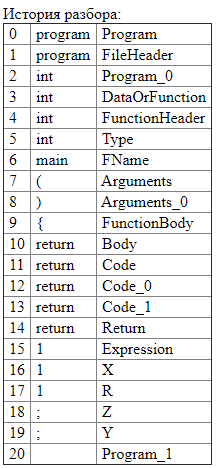
1. Запуск и инициализация. Очистить стек, прочитать первый символ входной цепочки, установить в качестве текущего состояния 0 и перейти к шагу 2.
2. Проверить, принадлежит ли очередной символ множеству выбора текущего состояния. Если да, то перейти к шагу 3, иначе - к шагу 6.
3. Если в клетке текущего состояния установлен флажок **а**, то прочитать следующий символ входной цепочки.
4. Если в клетке текущего состояния установлен флажок **s**, то поместить в стек номер текущего состояния, увеличенный на единицу.
5. Определение номера следующего состояния. Для этого прежде всего проверяется значение флажка **r** текущего состояния.
   1. Если флажок r установлен, то:
      1. Если стек не пуст, снять с верхушки стека номер состояния, установить его в качестве текущего и перейти к шагу 2;
      2. Если стек пуст – перейти к шагу 7.
   2. Если флажок r не установлен, то:
      1. Если текущим является состояние 1:
         1. Если стек пуст, те перейти к шагу 8.
         2. Если стек не пуст, перейти к шагу 7.
      2. Если текущим является любое другое состояние, то взять номер состояния из поля адреса перехода клетки текущего состояния. Установить в качестве текущего состояние с этим номером и вернуться к шагу 2.
6. Если в клетке текущего состояния установлен флажок е, то установить в качестве текущего состояния следующее (его номер вычисляется, как номер текущего состояния плюс единица) и вернуться к шагу 2, иначе – перейти к шагу 7.
7. Останов по ошибке.
8. Останов по окончанию разбора правильного предложения.

Конечный автомат со стековой памятью и одним состоянием



При запуске автомата (перед тактом 0) в стек заносятся символы ►и S, текущим входным символом устанавливается первый символ входной цепочки. Далее автомат, управляемый текущим входным символом и символом, находящимся на верхушке стека, реализует историю работы, показанную на рисунке выше, приводящую к останову на такте 32 по успешному восстановлению дерева грамматического разбора.

Процедурная реализацию рекурсивного спуска



Первый столбец – номер такта, второй столбец – очередной символ входного предложения, третий столбец – название активной функции.

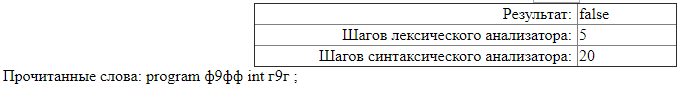
Считывая слова, рекурсивно спускаемся к терминалу, к которому относится данное слово, а затем возвращаемся обратно и повторяем для других слов.

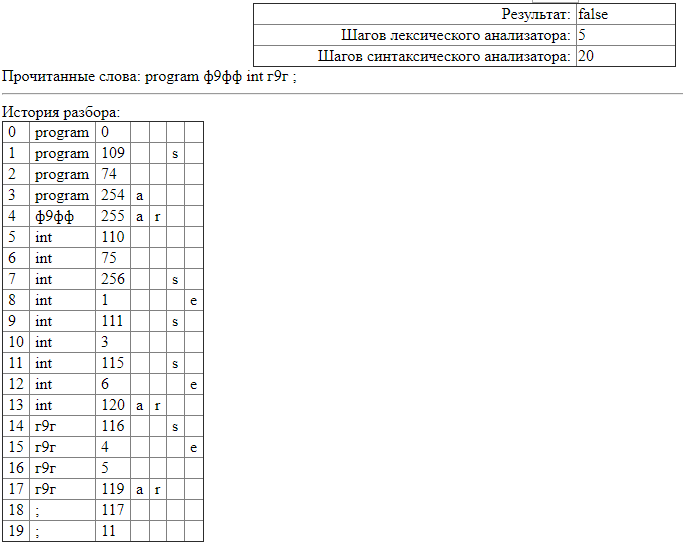
*Пример неправильной программы:*

program ф9фф

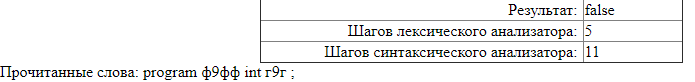
int г9г;

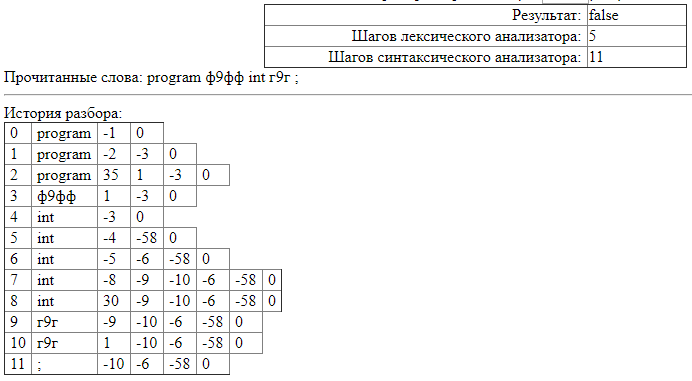
Конечный автомат со стековой памятью и несколькими состояниями:



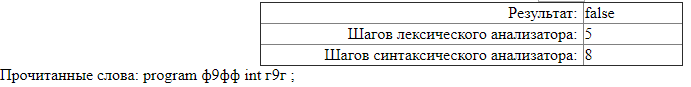


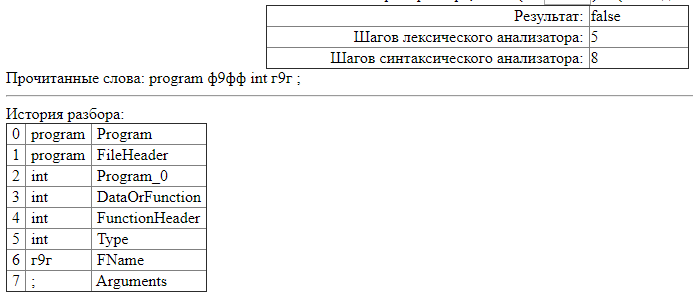
Конечный автомат со стековой памятью и одним состоянием





Процедурная реализацию рекурсивного спуска



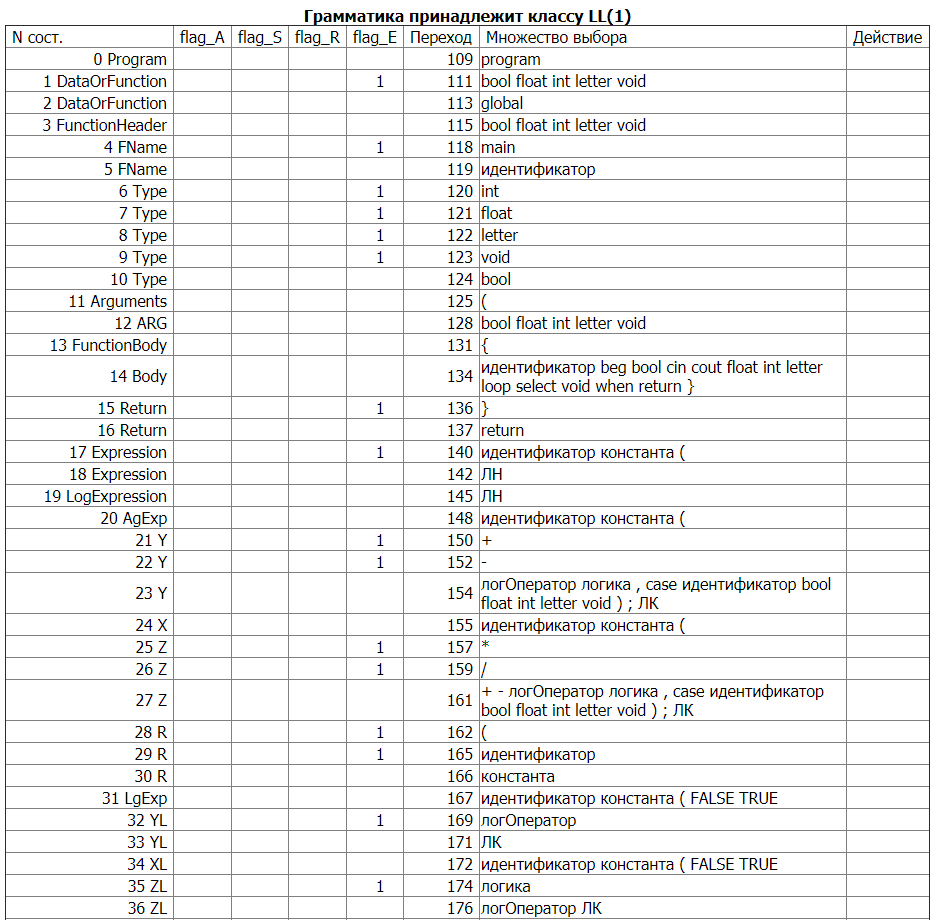


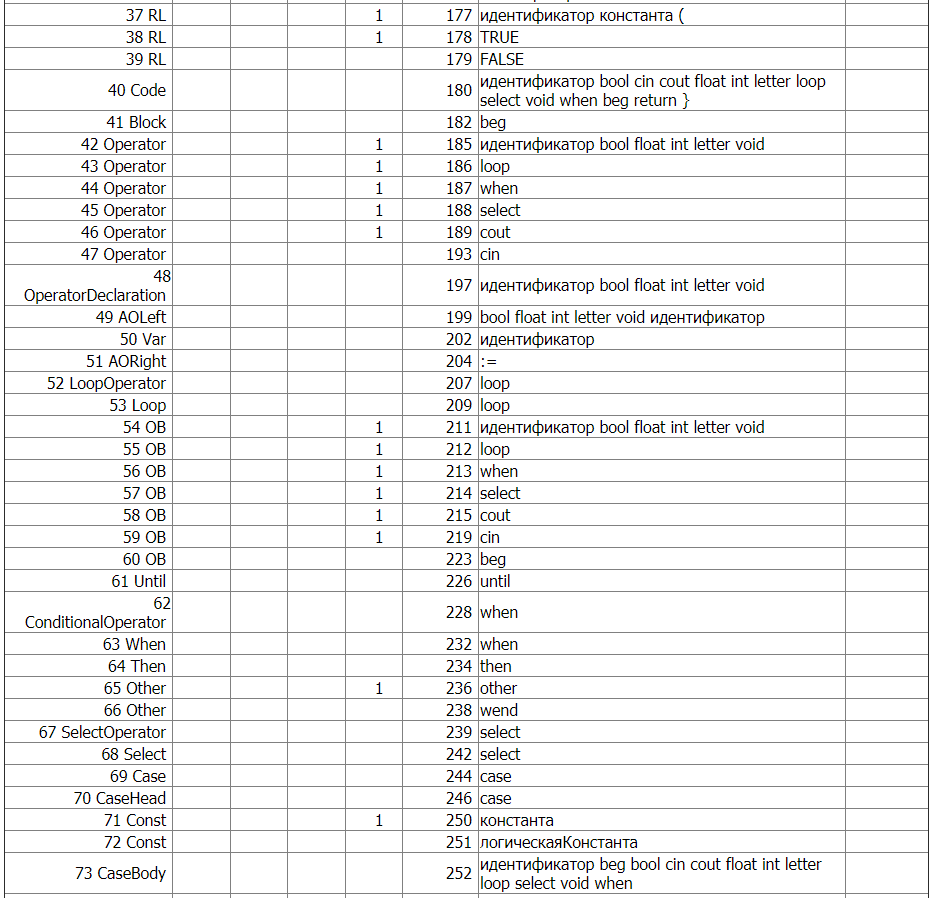
# **Выводы**

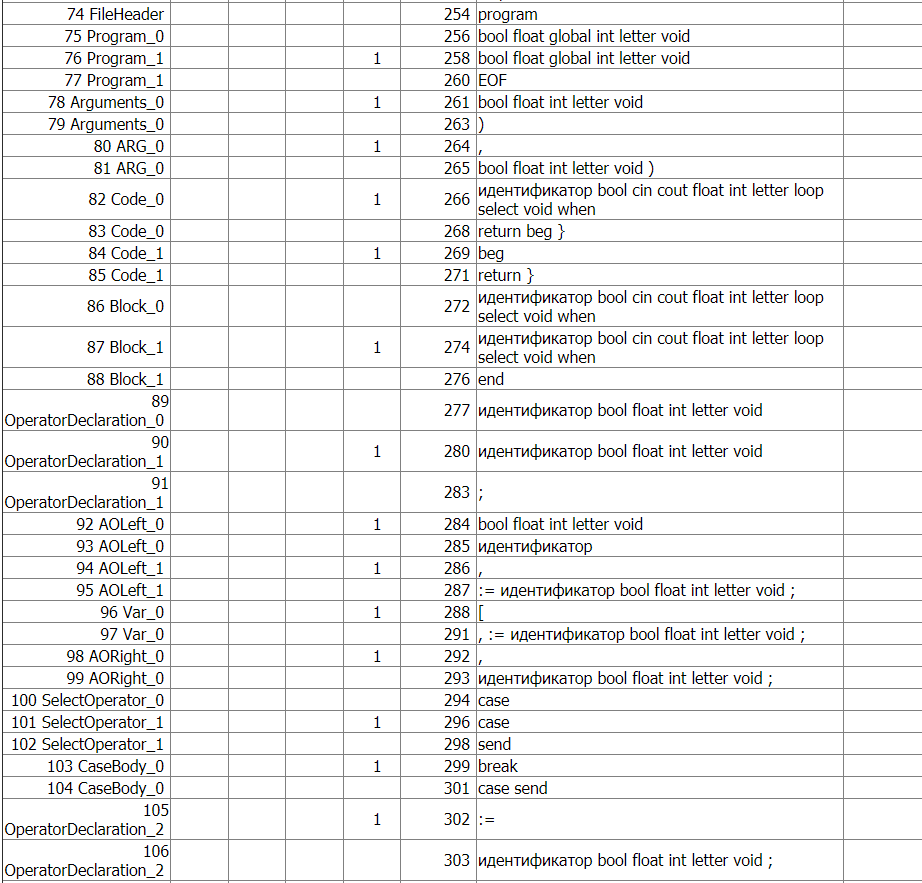
Были изучены основные идей и понятия нисходящих методов синтаксического анализа, выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобретены навыки построения процедурной и различных автоматных реализаций нисходящего анализа, исследовано поведения нисходящих синтаксических акцепторов.

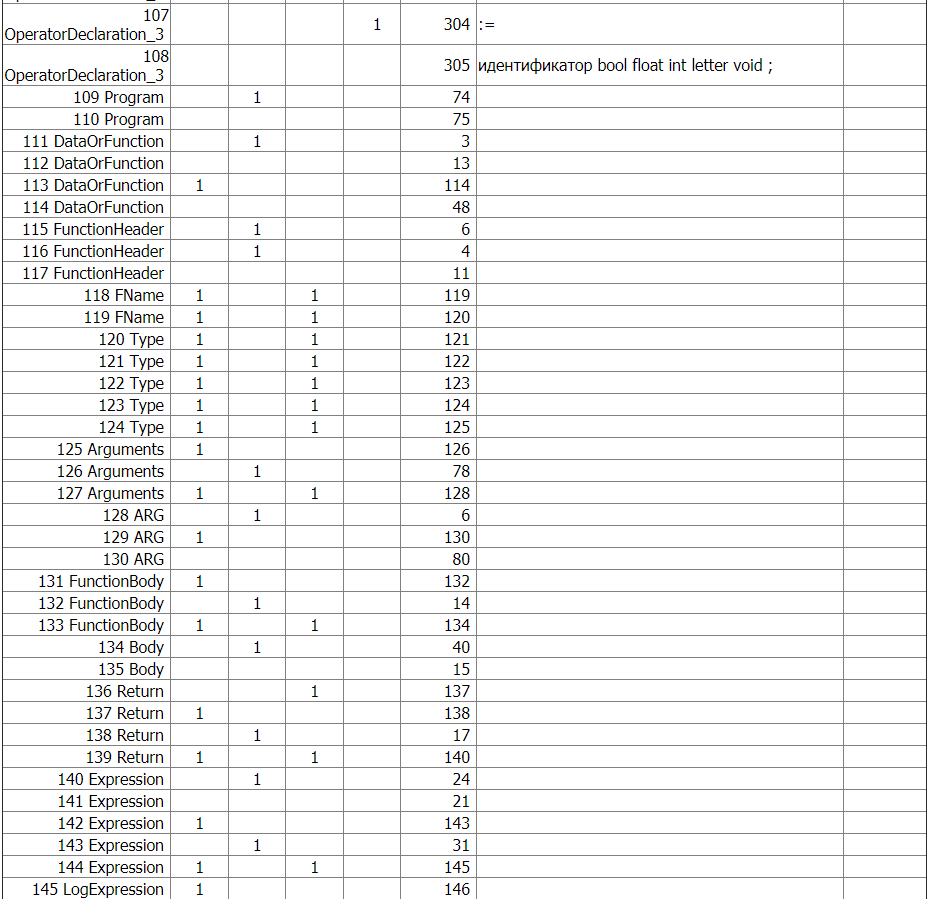
# **Приложение**

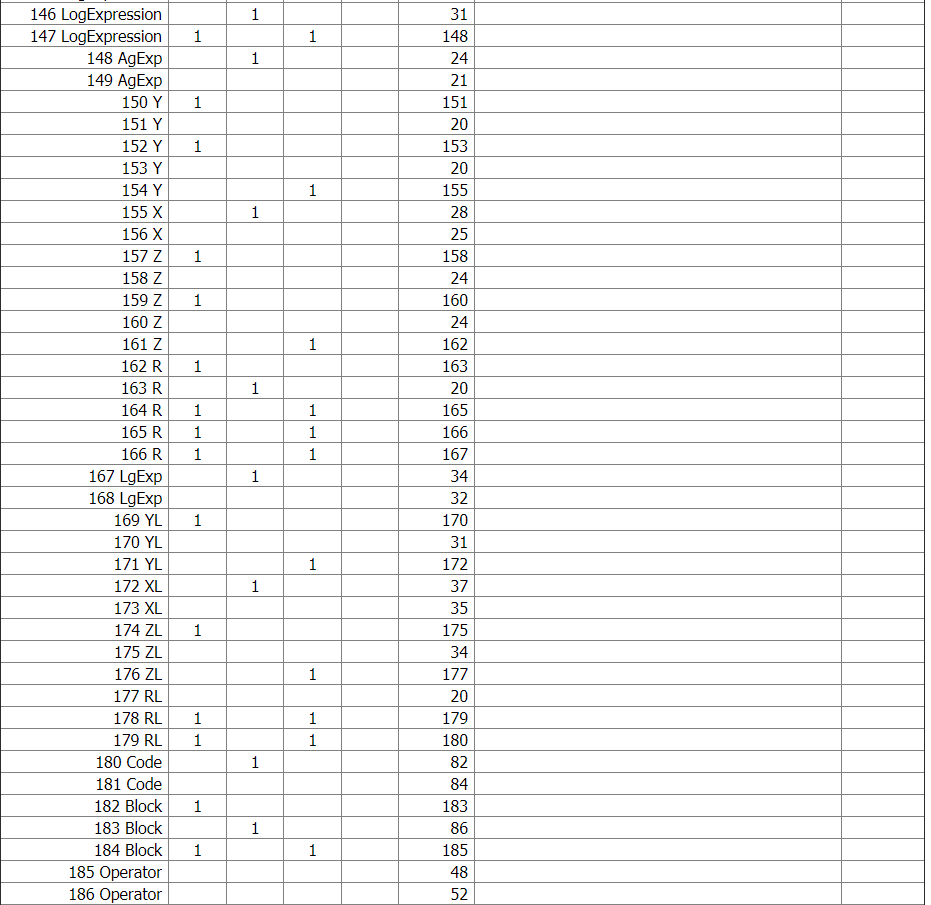
Приложение 1

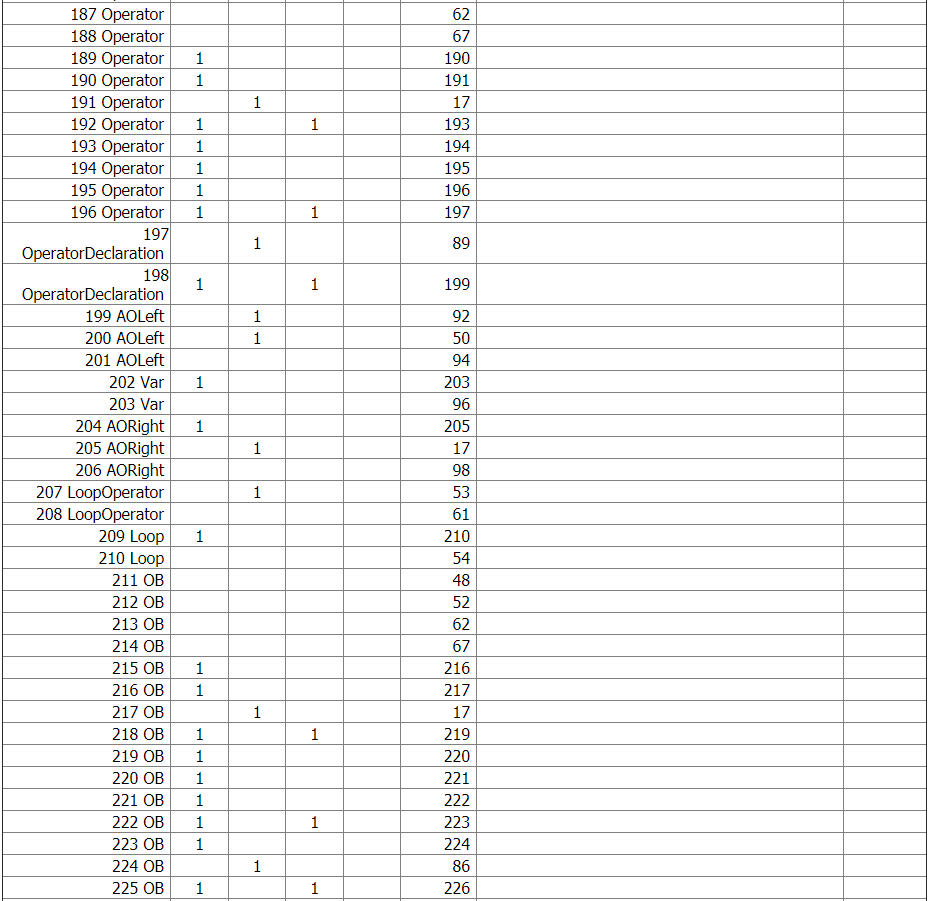


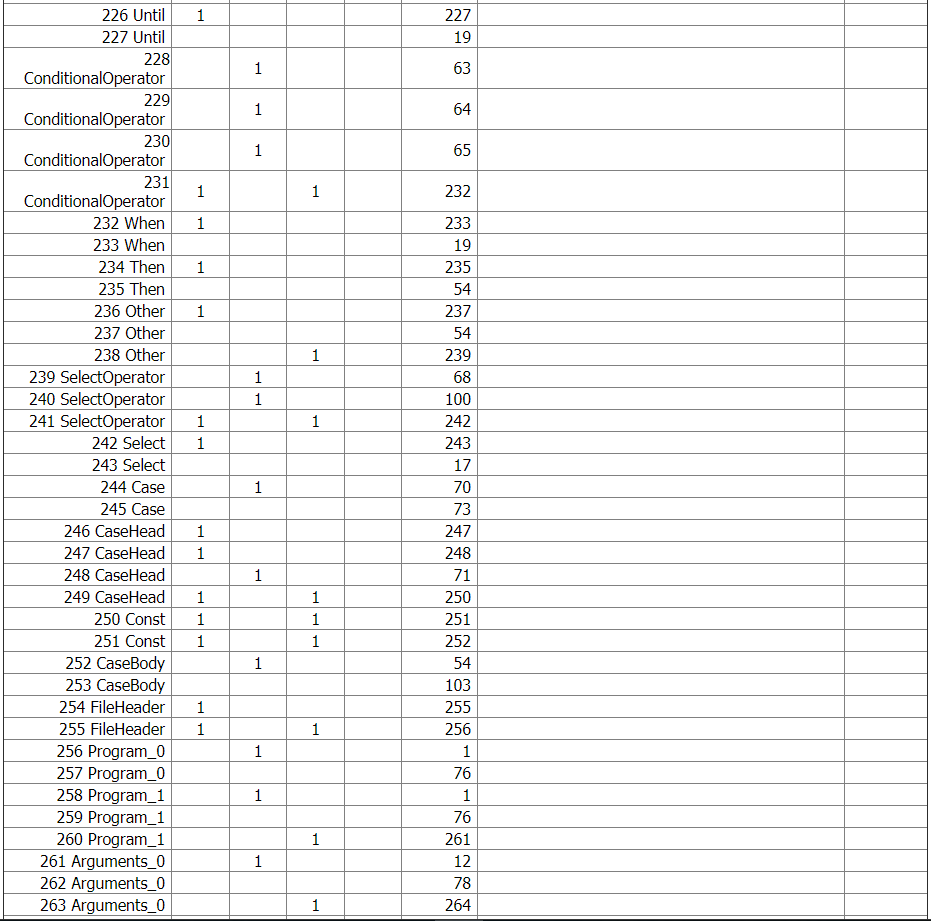


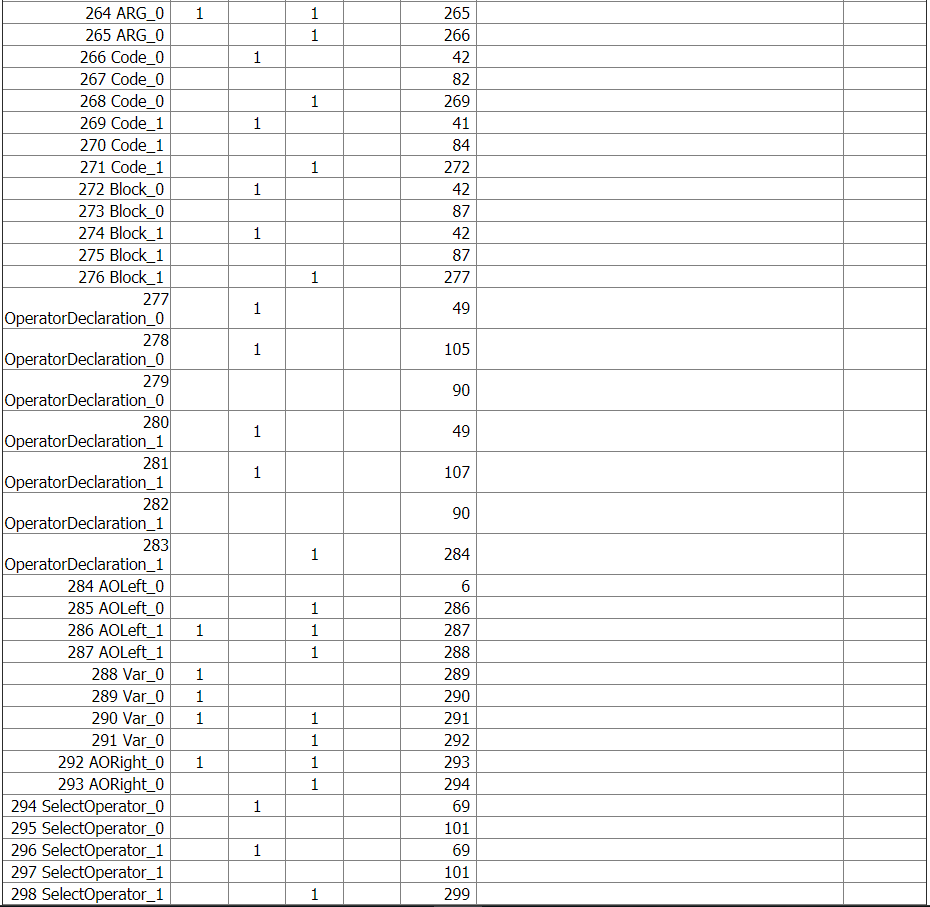


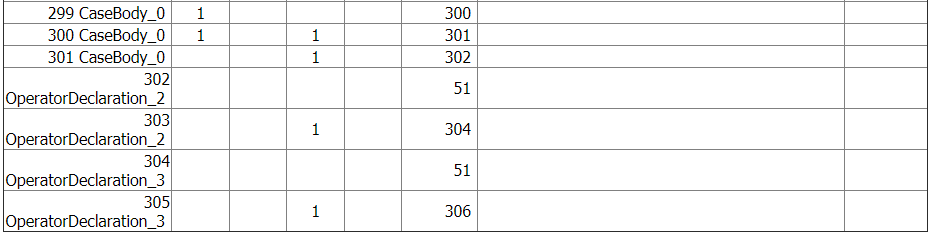












Приложение 2

//Построено 13.4.2020 17:26:51 по исходному файлу/шаблону lr1.xml/lexAsGraphSyntAsRD\_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

//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(0,43,rdr);

state = new FAstate(21);state.setArc(1, "T");state.setArc(2, "F");state.setArc(3, ",");state.setArc(4, "/");state.setArc(5, "<");state.setArc(6, ">");state.setArc(7, "Л");state.setArc(8, "|");state.setArc(9, "&");state.setArc(10, "=");state.setArc(11, "!");state.setArc(12, ":");state.setArc(13, ";");state.setArc(14, "(){}[]");state.setArc(15, "-+\*");state.setArc(16, " \n\r\u0009");state.setArc(17, "abcdefghijklmnopqrstuvwxyz\_");state.setArc(18, "0");state.setArc(19, "123456789");state.setArc(20, "\'");state.setArc(21, "абвгдежзийклмнопрстуфхцчшщъыьэюя");

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

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

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

state = new FAstate(1);state.setArc(23, "A");

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

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

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

state = new FAstate(2);state.setArc(-3, "");state.setArc(24, "/");

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

state = new FAstate(3);state.setArc(-4, "");state.setArc(25, "<");state.setArc(26, "=");

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

state = new FAstate(3);state.setArc(-4, "");state.setArc(27, ">");state.setArc(26, "=");

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

state = new FAstate(1);state.setArc(28, "НК");

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

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

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

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

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

state = new FAstate(2);state.setArc(-4, "");state.setArc(26, "=");

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

state = new FAstate(1);state.setArc(26, "=");

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

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

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

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

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

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

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

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

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

state = new FAstate(1);state.setArc(-7, "");

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

state = new FAstate(2);state.setArc(-8, "");state.setArc(17, "abcdefghijklmnopqrstuvwxyz\_");

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

state = new FAstate(5);state.setArc(-9, "");state.setArc(31, "0123");state.setArc(32, "456789");state.setArc(33, ".");state.setArc(34, "x");

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

state = new FAstate(3);state.setArc(-9, "");state.setArc(19, "0123456789");state.setArc(33, ".");

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

state = new FAstate(2);state.setArc(35, "");state.setArc(36, "\'");

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

state = new FAstate(1);state.setArc(37, "0123456789");

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

state = new FAstate(1);state.setArc(38, "U");

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

state = new FAstate(1);state.setArc(39, "L");

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

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

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

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

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

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

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

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

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

state = new FAstate(1);state.setArc(-13, "");

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

state = new FAstate(1);state.setArc(-14, "");

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

state = new FAstate(1);state.setArc(-15, "");

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

state = new FAstate(4);state.setArc(-9, "");state.setArc(31, "0123");state.setArc(32, "456789");state.setArc(33, ".");

this.lexAcceptors[0].setState(31,state);

state = new FAstate(2);state.setArc(32, "0123456789");state.setArc(33, ".");

this.lexAcceptors[0].setState(32,state);

state = new FAstate(2);state.setArc(-9, "");state.setArc(33, "0123456789");

this.lexAcceptors[0].setState(33,state);

state = new FAstate(1);state.setArc(40, "0123456789ABCDEF");

this.lexAcceptors[0].setState(34,state);

state = new FAstate(2);state.setArc(35, "");state.setArc(36, "\'");

this.lexAcceptors[0].setState(35,state);

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

this.lexAcceptors[0].setState(36,state);

state = new FAstate(2);state.setArc(37, "0123456789");state.setArc(41, "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя");

this.lexAcceptors[0].setState(37,state);

state = new FAstate(1);state.setArc(42, "E");

this.lexAcceptors[0].setState(38,state);

state = new FAstate(1);state.setArc(38, "S");

this.lexAcceptors[0].setState(39,state);

state = new FAstate(2);state.setArc(-9, "");state.setArc(40, "0123456789ABCDEF");

this.lexAcceptors[0].setState(40,state);

state = new FAstate(2);state.setArc(-16, "");state.setArc(41, "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя");

this.lexAcceptors[0].setState(41,state);

state = new FAstate(1);state.setArc(-17, "");

this.lexAcceptors[0].setState(42,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: //разделитель

break;

case -3: //знакОперации

break;

case -4: //логика

lexem[0] = 4;

break;

case -5: //ограничитель

break;

case -6: //скобки

break;

case -7: //пробелы

ignoreLastWord = true;break;

case -8: //ключевыеСлова

break;

case -9: //константа

lexem[0] = 2;

break;

case -10: //ComLineBegin

ignoreLastWord = true;

stack.push(this.lexAcceptor);

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

case -11: //вывод

lexem[0] = 6;

break;

case -12: //ввод

lexem[0] = 7;

break;

case -13: //ВыделениеЛогики

break;

case -14: //логОператор

lexem[0] = 3;

break;

case -15: //операторПрисваивания

break;

case -16: //идентификатор

lexem[0] = 1;

break;

case -17: //логическаяКонстанта

lexem[0] = 5;

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;

}

};

// вспомогательные классы синтаксического акцептора

function BitsSet(size){ //конструктор

var l = (size >> 5) + 1;

this.bs = new Array(l);

for(var i=0; i < l; i++)

this.bs[i]=0;

};

BitsSet.prototype = {

set: function(bitIndex){ //метод установки заданного бита в 1

var w;

var b;

if(bitIndex<0)

return; //ничего не делаем, если номер заданного бита отрицателен

w = bitIndex >> 5;

b = bitIndex & 31; //вычисляем индекс бита в слове

this.bs[w] |= 1<<b; //и устанавливаем его в 1

},

get: function(bitIndex){ //метод возращает значение заданного бита

var w;

var b;

if(bitIndex < 0)

return false; //ничего не делаем, если номер заданного бита отрицателен

w = bitIndex >> 5;

b = bitIndex & 31; //вычисляем индекс бита в слове

return ((this.bs[w]&(1<<b))==0?false:true); //и вернем его значение в логическом формате

}

};

var currentWord;

// синтаксический акцептор/анализатор

function Parser(reader){

this.currentLexem = [];

this.curWordIndex = 0;

this.la = new LexAnalyzer(reader);

this.cCnt = 0;

this.begS = 0;

this.endS = 0;

this.cntHist = 0;

this.stk = [];

this.selSet = new Array(109);

//далее для каждого правила образуется свое множество выбора и устанавливаются в 1 биты тех терминалов, которые входят в данное множество

//правила показаны в качестве комментария к оператору создания его множества выбора, имена терминалов в комментариях к оператору установки бита

this.selSet[0]=new BitsSet(48);// Множество выбора правила Program : FileHeader Program\_0

this.selSet[0].set(35); // program

this.selSet[1]=new BitsSet(48);// Множество выбора правила DataOrFunction : FunctionHeader FunctionBody

this.selSet[1].set(22); // bool

this.selSet[1].set(28); // float

this.selSet[1].set(30); // int

this.selSet[1].set(31); // letter

this.selSet[1].set(41); // void

this.selSet[2]=new BitsSet(48);// Множество выбора правила DataOrFunction : global OperatorDeclaration

this.selSet[2].set(29); // global

this.selSet[3]=new BitsSet(48);// Множество выбора правила FunctionHeader : Type FName Arguments

this.selSet[3].set(22); // bool

this.selSet[3].set(28); // float

this.selSet[3].set(30); // int

this.selSet[3].set(31); // letter

this.selSet[3].set(41); // void

this.selSet[4]=new BitsSet(48);// Множество выбора правила FName : main

this.selSet[4].set(33); // main

this.selSet[5]=new BitsSet(48);// Множество выбора правила FName : идентификатор

this.selSet[5].set(1); // идентификатор

this.selSet[6]=new BitsSet(48);// Множество выбора правила Type : int

this.selSet[6].set(30); // int

this.selSet[7]=new BitsSet(48);// Множество выбора правила Type : float

this.selSet[7].set(28); // float

this.selSet[8]=new BitsSet(48);// Множество выбора правила Type : letter

this.selSet[8].set(31); // letter

this.selSet[9]=new BitsSet(48);// Множество выбора правила Type : void

this.selSet[9].set(41); // void

this.selSet[10]=new BitsSet(48);// Множество выбора правила Type : bool

this.selSet[10].set(22); // bool

this.selSet[11]=new BitsSet(48);// Множество выбора правила Arguments : ( Arguments\_0 )

this.selSet[11].set(8); // (

this.selSet[12]=new BitsSet(48);// Множество выбора правила ARG : Type идентификатор ARG\_0

this.selSet[12].set(22); // bool

this.selSet[12].set(28); // float

this.selSet[12].set(30); // int

this.selSet[12].set(31); // letter

this.selSet[12].set(41); // void

this.selSet[13]=new BitsSet(48);// Множество выбора правила FunctionBody : { Body }

this.selSet[13].set(44); // {

this.selSet[14]=new BitsSet(48);// Множество выбора правила Body : Code Return

this.selSet[14].set(1); // идентификатор

this.selSet[14].set(21); // beg

this.selSet[14].set(22); // bool

this.selSet[14].set(25); // cin

this.selSet[14].set(26); // cout

this.selSet[14].set(28); // float

this.selSet[14].set(30); // int

this.selSet[14].set(31); // letter

this.selSet[14].set(32); // loop

this.selSet[14].set(36); // return

this.selSet[14].set(37); // select

this.selSet[14].set(41); // void

this.selSet[14].set(43); // when

this.selSet[14].set(45); // }

this.selSet[15]=new BitsSet(48);// Множество выбора правила Return :

this.selSet[15].set(45); // }

this.selSet[16]=new BitsSet(48);// Множество выбора правила Return : return Expression ;

this.selSet[16].set(36); // return

this.selSet[17]=new BitsSet(48);// Множество выбора правила Expression : X Y

this.selSet[17].set(1); // идентификатор

this.selSet[17].set(2); // константа

this.selSet[17].set(8); // (

this.selSet[18]=new BitsSet(48);// Множество выбора правила Expression : ЛН LgExp ЛК

this.selSet[18].set(47); // ЛН

this.selSet[19]=new BitsSet(48);// Множество выбора правила LogExpression : ЛН LgExp ЛК

this.selSet[19].set(47); // ЛН

this.selSet[20]=new BitsSet(48);// Множество выбора правила AgExp : X Y

this.selSet[20].set(1); // идентификатор

this.selSet[20].set(2); // константа

this.selSet[20].set(8); // (

this.selSet[21]=new BitsSet(48);// Множество выбора правила Y : + AgExp

this.selSet[21].set(11); // +

this.selSet[22]=new BitsSet(48);// Множество выбора правила Y : - AgExp

this.selSet[22].set(13); // -

this.selSet[23]=new BitsSet(48);// Множество выбора правила Y :

this.selSet[23].set(1); // идентификатор

this.selSet[23].set(3); // логОператор

this.selSet[23].set(4); // логика

this.selSet[23].set(9); // )

this.selSet[23].set(12); // ,

this.selSet[23].set(16); // ;

this.selSet[23].set(22); // bool

this.selSet[23].set(24); // case

this.selSet[23].set(28); // float

this.selSet[23].set(30); // int

this.selSet[23].set(31); // letter

this.selSet[23].set(41); // void

this.selSet[23].set(46); // ЛК

this.selSet[24]=new BitsSet(48);// Множество выбора правила X : R Z

this.selSet[24].set(1); // идентификатор

this.selSet[24].set(2); // константа

this.selSet[24].set(8); // (

this.selSet[25]=new BitsSet(48);// Множество выбора правила Z : \* X

this.selSet[25].set(10); // \*

this.selSet[26]=new BitsSet(48);// Множество выбора правила Z : / X

this.selSet[26].set(14); // /

this.selSet[27]=new BitsSet(48);// Множество выбора правила Z :

this.selSet[27].set(1); // идентификатор

this.selSet[27].set(3); // логОператор

this.selSet[27].set(4); // логика

this.selSet[27].set(9); // )

this.selSet[27].set(11); // +

this.selSet[27].set(12); // ,

this.selSet[27].set(13); // -

this.selSet[27].set(16); // ;

this.selSet[27].set(22); // bool

this.selSet[27].set(24); // case

this.selSet[27].set(28); // float

this.selSet[27].set(30); // int

this.selSet[27].set(31); // letter

this.selSet[27].set(41); // void

this.selSet[27].set(46); // ЛК

this.selSet[28]=new BitsSet(48);// Множество выбора правила R : ( AgExp )

this.selSet[28].set(8); // (

this.selSet[29]=new BitsSet(48);// Множество выбора правила R : идентификатор

this.selSet[29].set(1); // идентификатор

this.selSet[30]=new BitsSet(48);// Множество выбора правила R : константа

this.selSet[30].set(2); // константа

this.selSet[31]=new BitsSet(48);// Множество выбора правила LgExp : XL YL

this.selSet[31].set(1); // идентификатор

this.selSet[31].set(2); // константа

this.selSet[31].set(8); // (

this.selSet[31].set(17); // FALSE

this.selSet[31].set(18); // TRUE

this.selSet[32]=new BitsSet(48);// Множество выбора правила YL : логОператор LgExp

this.selSet[32].set(3); // логОператор

this.selSet[33]=new BitsSet(48);// Множество выбора правила YL :

this.selSet[33].set(46); // ЛК

this.selSet[34]=new BitsSet(48);// Множество выбора правила XL : RL ZL

this.selSet[34].set(1); // идентификатор

this.selSet[34].set(2); // константа

this.selSet[34].set(8); // (

this.selSet[34].set(17); // FALSE

this.selSet[34].set(18); // TRUE

this.selSet[35]=new BitsSet(48);// Множество выбора правила ZL : логика XL

this.selSet[35].set(4); // логика

this.selSet[36]=new BitsSet(48);// Множество выбора правила ZL :

this.selSet[36].set(3); // логОператор

this.selSet[36].set(46); // ЛК

this.selSet[37]=new BitsSet(48);// Множество выбора правила RL : AgExp

this.selSet[37].set(1); // идентификатор

this.selSet[37].set(2); // константа

this.selSet[37].set(8); // (

this.selSet[38]=new BitsSet(48);// Множество выбора правила RL : TRUE

this.selSet[38].set(18); // TRUE

this.selSet[39]=new BitsSet(48);// Множество выбора правила RL : FALSE

this.selSet[39].set(17); // FALSE

this.selSet[40]=new BitsSet(48);// Множество выбора правила Code : Code\_0 Code\_1

this.selSet[40].set(1); // идентификатор

this.selSet[40].set(21); // beg

this.selSet[40].set(22); // bool

this.selSet[40].set(25); // cin

this.selSet[40].set(26); // cout

this.selSet[40].set(28); // float

this.selSet[40].set(30); // int

this.selSet[40].set(31); // letter

this.selSet[40].set(32); // loop

this.selSet[40].set(36); // return

this.selSet[40].set(37); // select

this.selSet[40].set(41); // void

this.selSet[40].set(43); // when

this.selSet[40].set(45); // }

this.selSet[41]=new BitsSet(48);// Множество выбора правила Block : beg Block\_0 end

this.selSet[41].set(21); // beg

this.selSet[42]=new BitsSet(48);// Множество выбора правила Operator : OperatorDeclaration

this.selSet[42].set(1); // идентификатор

this.selSet[42].set(22); // bool

this.selSet[42].set(28); // float

this.selSet[42].set(30); // int

this.selSet[42].set(31); // letter

this.selSet[42].set(41); // void

this.selSet[43]=new BitsSet(48);// Множество выбора правила Operator : LoopOperator

this.selSet[43].set(32); // loop

this.selSet[44]=new BitsSet(48);// Множество выбора правила Operator : ConditionalOperator

this.selSet[44].set(43); // when

this.selSet[45]=new BitsSet(48);// Множество выбора правила Operator : SelectOperator

this.selSet[45].set(37); // select

this.selSet[46]=new BitsSet(48);// Множество выбора правила Operator : cout вывод Expression ;

this.selSet[46].set(26); // cout

this.selSet[47]=new BitsSet(48);// Множество выбора правила Operator : cin ввод идентификатор ;

this.selSet[47].set(25); // cin

this.selSet[48]=new BitsSet(48);// Множество выбора правила OperatorDeclaration : OperatorDeclaration\_0 ;

this.selSet[48].set(1); // идентификатор

this.selSet[48].set(22); // bool

this.selSet[48].set(28); // float

this.selSet[48].set(30); // int

this.selSet[48].set(31); // letter

this.selSet[48].set(41); // void

this.selSet[49]=new BitsSet(48);// Множество выбора правила AOLeft : AOLeft\_0 Var AOLeft\_1

this.selSet[49].set(1); // идентификатор

this.selSet[49].set(22); // bool

this.selSet[49].set(28); // float

this.selSet[49].set(30); // int

this.selSet[49].set(31); // letter

this.selSet[49].set(41); // void

this.selSet[50]=new BitsSet(48);// Множество выбора правила Var : идентификатор Var\_0

this.selSet[50].set(1); // идентификатор

this.selSet[51]=new BitsSet(48);// Множество выбора правила AORight : := Expression AORight\_0

this.selSet[51].set(15); // :=

this.selSet[52]=new BitsSet(48);// Множество выбора правила LoopOperator : Loop Until

this.selSet[52].set(32); // loop

this.selSet[53]=new BitsSet(48);// Множество выбора правила Loop : loop OB

this.selSet[53].set(32); // loop

this.selSet[54]=new BitsSet(48);// Множество выбора правила OB : OperatorDeclaration

this.selSet[54].set(1); // идентификатор

this.selSet[54].set(22); // bool

this.selSet[54].set(28); // float

this.selSet[54].set(30); // int

this.selSet[54].set(31); // letter

this.selSet[54].set(41); // void

this.selSet[55]=new BitsSet(48);// Множество выбора правила OB : LoopOperator

this.selSet[55].set(32); // loop

this.selSet[56]=new BitsSet(48);// Множество выбора правила OB : ConditionalOperator

this.selSet[56].set(43); // when

this.selSet[57]=new BitsSet(48);// Множество выбора правила OB : SelectOperator

this.selSet[57].set(37); // select

this.selSet[58]=new BitsSet(48);// Множество выбора правила OB : cout вывод Expression ;

this.selSet[58].set(26); // cout

this.selSet[59]=new BitsSet(48);// Множество выбора правила OB : cin ввод идентификатор ;

this.selSet[59].set(25); // cin

this.selSet[60]=new BitsSet(48);// Множество выбора правила OB : beg Block\_0 end

this.selSet[60].set(21); // beg

this.selSet[61]=new BitsSet(48);// Множество выбора правила Until : until LogExpression

this.selSet[61].set(40); // until

this.selSet[62]=new BitsSet(48);// Множество выбора правила ConditionalOperator : When Then Other wend

this.selSet[62].set(43); // when

this.selSet[63]=new BitsSet(48);// Множество выбора правила When : when LogExpression

this.selSet[63].set(43); // when

this.selSet[64]=new BitsSet(48);// Множество выбора правила Then : then OB

this.selSet[64].set(39); // then

this.selSet[65]=new BitsSet(48);// Множество выбора правила Other : other OB

this.selSet[65].set(34); // other

this.selSet[66]=new BitsSet(48);// Множество выбора правила Other :

this.selSet[66].set(42); // wend

this.selSet[67]=new BitsSet(48);// Множество выбора правила SelectOperator : Select SelectOperator\_0 send

this.selSet[67].set(37); // select

this.selSet[68]=new BitsSet(48);// Множество выбора правила Select : select Expression

this.selSet[68].set(37); // select

this.selSet[69]=new BitsSet(48);// Множество выбора правила Case : CaseHead CaseBody

this.selSet[69].set(24); // case

this.selSet[70]=new BitsSet(48);// Множество выбора правила CaseHead : case ( Const )

this.selSet[70].set(24); // case

this.selSet[71]=new BitsSet(48);// Множество выбора правила Const : константа

this.selSet[71].set(2); // константа

this.selSet[72]=new BitsSet(48);// Множество выбора правила Const : логическаяКонстанта

this.selSet[72].set(5); // логическаяКонстанта

this.selSet[73]=new BitsSet(48);// Множество выбора правила CaseBody : OB CaseBody\_0

this.selSet[73].set(1); // идентификатор

this.selSet[73].set(21); // beg

this.selSet[73].set(22); // bool

this.selSet[73].set(25); // cin

this.selSet[73].set(26); // cout

this.selSet[73].set(28); // float

this.selSet[73].set(30); // int

this.selSet[73].set(31); // letter

this.selSet[73].set(32); // loop

this.selSet[73].set(37); // select

this.selSet[73].set(41); // void

this.selSet[73].set(43); // when

this.selSet[74]=new BitsSet(48);// Множество выбора правила FileHeader : program идентификатор

this.selSet[74].set(35); // program

this.selSet[75]=new BitsSet(48);// Множество выбора правила Program\_0 : DataOrFunction Program\_1

this.selSet[75].set(22); // bool

this.selSet[75].set(28); // float

this.selSet[75].set(29); // global

this.selSet[75].set(30); // int

this.selSet[75].set(31); // letter

this.selSet[75].set(41); // void

this.selSet[76]=new BitsSet(48);// Множество выбора правила Program\_1 : DataOrFunction Program\_1

this.selSet[76].set(22); // bool

this.selSet[76].set(28); // float

this.selSet[76].set(29); // global

this.selSet[76].set(30); // int

this.selSet[76].set(31); // letter

this.selSet[76].set(41); // void

this.selSet[77]=new BitsSet(48);// Множество выбора правила Program\_1 :

this.selSet[77].set(0); // EOF

this.selSet[78]=new BitsSet(48);// Множество выбора правила Arguments\_0 : ARG Arguments\_0

this.selSet[78].set(22); // bool

this.selSet[78].set(28); // float

this.selSet[78].set(30); // int

this.selSet[78].set(31); // letter

this.selSet[78].set(41); // void

this.selSet[79]=new BitsSet(48);// Множество выбора правила Arguments\_0 :

this.selSet[79].set(9); // )

this.selSet[80]=new BitsSet(48);// Множество выбора правила ARG\_0 : ,

this.selSet[80].set(12); // ,

this.selSet[81]=new BitsSet(48);// Множество выбора правила ARG\_0 :

this.selSet[81].set(9); // )

this.selSet[81].set(22); // bool

this.selSet[81].set(28); // float

this.selSet[81].set(30); // int

this.selSet[81].set(31); // letter

this.selSet[81].set(41); // void

this.selSet[82]=new BitsSet(48);// Множество выбора правила Code\_0 : Operator Code\_0

this.selSet[82].set(1); // идентификатор

this.selSet[82].set(22); // bool

this.selSet[82].set(25); // cin

this.selSet[82].set(26); // cout

this.selSet[82].set(28); // float

this.selSet[82].set(30); // int

this.selSet[82].set(31); // letter

this.selSet[82].set(32); // loop

this.selSet[82].set(37); // select

this.selSet[82].set(41); // void

this.selSet[82].set(43); // when

this.selSet[83]=new BitsSet(48);// Множество выбора правила Code\_0 :

this.selSet[83].set(21); // beg

this.selSet[83].set(36); // return

this.selSet[83].set(45); // }

this.selSet[84]=new BitsSet(48);// Множество выбора правила Code\_1 : Block Code\_1

this.selSet[84].set(21); // beg

this.selSet[85]=new BitsSet(48);// Множество выбора правила Code\_1 :

this.selSet[85].set(36); // return

this.selSet[85].set(45); // }

this.selSet[86]=new BitsSet(48);// Множество выбора правила Block\_0 : Operator Block\_1

this.selSet[86].set(1); // идентификатор

this.selSet[86].set(22); // bool

this.selSet[86].set(25); // cin

this.selSet[86].set(26); // cout

this.selSet[86].set(28); // float

this.selSet[86].set(30); // int

this.selSet[86].set(31); // letter

this.selSet[86].set(32); // loop

this.selSet[86].set(37); // select

this.selSet[86].set(41); // void

this.selSet[86].set(43); // when

this.selSet[87]=new BitsSet(48);// Множество выбора правила Block\_1 : Operator Block\_1

this.selSet[87].set(1); // идентификатор

this.selSet[87].set(22); // bool

this.selSet[87].set(25); // cin

this.selSet[87].set(26); // cout

this.selSet[87].set(28); // float

this.selSet[87].set(30); // int

this.selSet[87].set(31); // letter

this.selSet[87].set(32); // loop

this.selSet[87].set(37); // select

this.selSet[87].set(41); // void

this.selSet[87].set(43); // when

this.selSet[88]=new BitsSet(48);// Множество выбора правила Block\_1 :

this.selSet[88].set(27); // end

this.selSet[89]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_0 : AOLeft OperatorDeclaration\_2 OperatorDeclaration\_1

this.selSet[89].set(1); // идентификатор

this.selSet[89].set(22); // bool

this.selSet[89].set(28); // float

this.selSet[89].set(30); // int

this.selSet[89].set(31); // letter

this.selSet[89].set(41); // void

this.selSet[90]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_1 : AOLeft OperatorDeclaration\_3 OperatorDeclaration\_1

this.selSet[90].set(1); // идентификатор

this.selSet[90].set(22); // bool

this.selSet[90].set(28); // float

this.selSet[90].set(30); // int

this.selSet[90].set(31); // letter

this.selSet[90].set(41); // void

this.selSet[91]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_1 :

this.selSet[91].set(16); // ;

this.selSet[92]=new BitsSet(48);// Множество выбора правила AOLeft\_0 : Type

this.selSet[92].set(22); // bool

this.selSet[92].set(28); // float

this.selSet[92].set(30); // int

this.selSet[92].set(31); // letter

this.selSet[92].set(41); // void

this.selSet[93]=new BitsSet(48);// Множество выбора правила AOLeft\_0 :

this.selSet[93].set(1); // идентификатор

this.selSet[94]=new BitsSet(48);// Множество выбора правила AOLeft\_1 : ,

this.selSet[94].set(12); // ,

this.selSet[95]=new BitsSet(48);// Множество выбора правила AOLeft\_1 :

this.selSet[95].set(1); // идентификатор

this.selSet[95].set(15); // :=

this.selSet[95].set(16); // ;

this.selSet[95].set(22); // bool

this.selSet[95].set(28); // float

this.selSet[95].set(30); // int

this.selSet[95].set(31); // letter

this.selSet[95].set(41); // void

this.selSet[96]=new BitsSet(48);// Множество выбора правила Var\_0 : [ константа ]

this.selSet[96].set(19); // [

this.selSet[97]=new BitsSet(48);// Множество выбора правила Var\_0 :

this.selSet[97].set(1); // идентификатор

this.selSet[97].set(12); // ,

this.selSet[97].set(15); // :=

this.selSet[97].set(16); // ;

this.selSet[97].set(22); // bool

this.selSet[97].set(28); // float

this.selSet[97].set(30); // int

this.selSet[97].set(31); // letter

this.selSet[97].set(41); // void

this.selSet[98]=new BitsSet(48);// Множество выбора правила AORight\_0 : ,

this.selSet[98].set(12); // ,

this.selSet[99]=new BitsSet(48);// Множество выбора правила AORight\_0 :

this.selSet[99].set(1); // идентификатор

this.selSet[99].set(16); // ;

this.selSet[99].set(22); // bool

this.selSet[99].set(28); // float

this.selSet[99].set(30); // int

this.selSet[99].set(31); // letter

this.selSet[99].set(41); // void

this.selSet[100]=new BitsSet(48);// Множество выбора правила SelectOperator\_0 : Case SelectOperator\_1

this.selSet[100].set(24); // case

this.selSet[101]=new BitsSet(48);// Множество выбора правила SelectOperator\_1 : Case SelectOperator\_1

this.selSet[101].set(24); // case

this.selSet[102]=new BitsSet(48);// Множество выбора правила SelectOperator\_1 :

this.selSet[102].set(38); // send

this.selSet[103]=new BitsSet(48);// Множество выбора правила CaseBody\_0 : break ;

this.selSet[103].set(23); // break

this.selSet[104]=new BitsSet(48);// Множество выбора правила CaseBody\_0 :

this.selSet[104].set(24); // case

this.selSet[104].set(38); // send

this.selSet[105]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_2 : AORight

this.selSet[105].set(15); // :=

this.selSet[106]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_2 :

this.selSet[106].set(1); // идентификатор

this.selSet[106].set(16); // ;

this.selSet[106].set(22); // bool

this.selSet[106].set(28); // float

this.selSet[106].set(30); // int

this.selSet[106].set(31); // letter

this.selSet[106].set(41); // void

this.selSet[107]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_3 : AORight

this.selSet[107].set(15); // :=

this.selSet[108]=new BitsSet(48);// Множество выбора правила OperatorDeclaration\_3 :

this.selSet[108].set(1); // идентификатор

this.selSet[108].set(16); // ;

this.selSet[108].set(22); // bool

this.selSet[108].set(28); // float

this.selSet[108].set(30); // int

this.selSet[108].set(31); // letter

this.selSet[108].set(41); // void

this.words = [ "(", ")", "\*", "+", ",", "-", "/", ":=", ";", "FALSE", "TRUE", "[", "]", "beg", "bool", "break", "case", "cin", "cout", "end", "float", "global", "int", "letter", "loop", "main", "other", "program", "return", "select", "send", "then", "until", "void", "wend", "when", "{", "}", "ЛК", "ЛН",""];

};

Parser.prototype = {

// вызов лексического анализатора и преобразование лексемы в код терминального символа

getWordIndex: function(){ //эта функция вызывает лексический анализатор, получает лексему и пытается найти обнаруженное слово

// в массиве строк литер words. При удаче - возвращает индекс этой строки, иначе - индекс группы слов, сформированный лексическим анализатором.

var i;

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

currentWord = this.currentLexem[1];

for(var i=0; i < this.words.length - 1; i++) //просмотрим слова, определенные в грамматике как строки литер

if(this.words[i] == this.currentLexem[1]) //если входное слово найдено в массиве строк литер

return i+8; //вычислим и вернем его индекс

return this.currentLexem[0]; //вернем индекс группы слов

},

getInfo: function(i){

switch (i) {

case 0:

return lCnt;

case 1:

return this.cCnt;

case 2:

return this.wList;

case 3:

if((this.cntHist>0) && (this.pHist.length < this.cntHist)){

this.pHist.splice(this.begS,1);

this.cntHist = 0;

}

return this.pHist;

}

return null;

},

putHist: function(n){

this.pHist.push([this.cCnt, this.currentLexem[1], n]);

if(this.pHist.length == this.begS)

this.pHist.push(["..."]);

if(this.pHist.length > this.cntHist + 1)

this.pHist.splice(this.begS+1, 1);

},

// собственно восстановление дерева грамматического разбора

parse: function(begS, endS){

this.cntHist = begS + endS;

this.begS = begS;

this.endS = endS;

lCnt = 0;

this.cCnt = 0;

this.wList = [];

this.pHist = [];

this.curWordIndex=this.getWordIndex(); //возьмем индекс первого терминала

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

if(this.Program()) //вызовем метод, разбирающий начальный нетерминал

if(this.curWordIndex==0)

return true; //если этот метод возвратил истину и весь входной поток прочитан - последовательность слов правильна, возвращаем 0

return false; //если что-то не так - вернем код ошибки 1

},

//все далее следующие методы:

// - имеют имя нетерминала, разбором которого каждый из них занимается

// - возвращают логическое значение

// - содержат ровно столько последовательно выполняющихся блоков, сколько правил в грамматике есть для данного нетерминала

// - каждый блок выполняется только при условии, что текущий терминал принадлежит множеству выбора донного правила

// - внутри блока организована обработка правой части правила последовательно символ за символом слева направо;

// в зависимости от того, чем является каждый очередной символ, делается следующее:

// - для нетерминала - вызывается метод, имеющий его имя и, если от него получена ложь, то ложь возвращается выше;

// - для терминала - его индекс сравнивается с индексом текущего терминала, при несовпадении возвращается ложь (эта проверка не выполняется, если терминал есть первый символ правила),

// затем из входного потока читается следующий терминал; терминал указывается в примечании к оператору сравнения;

// - для действия - оно выполняется (пустая правая часть правила приравнивается к действию, для таких правил формируется {});

// если выполнена обработка всех символов правой части (и не была возвращена ложь), то возвращается истина

// - если не будет выполнен ни один блок (текущий терминал не принадлежит множеству выбора ни одного правила) - возвращается значение ложь

Program: function(){

if(this.cntHist > 0)

this.putHist("Program");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[0].get(this.curWordIndex)){

if(!this.FileHeader())

return false;

return this.Program\_0();

}

return false;

},

DataOrFunction: function(){

if(this.cntHist > 0)

this.putHist("DataOrFunction");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[1].get(this.curWordIndex)){

if(!this.FunctionHeader())

return false;

return this.FunctionBody();

}

if((this.curWordIndex >= 0) && this.selSet[2].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.OperatorDeclaration();

}

return false;

},

FunctionHeader: function(){

if(this.cntHist > 0)

this.putHist("FunctionHeader");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[3].get(this.curWordIndex)){

if(!this.Type())

return false;

if(!this.FName())

return false;

return this.Arguments();

}

return false;

},

FName: function(){

if(this.cntHist > 0)

this.putHist("FName");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[4].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[5].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Type: function(){

if(this.cntHist > 0)

this.putHist("Type");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[6].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[7].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[8].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[9].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[10].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Arguments: function(){

if(this.cntHist > 0)

this.putHist("Arguments");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[11].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Arguments\_0())

return false;

if(this.curWordIndex != 9) // )

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

ARG: function(){

if(this.cntHist > 0)

this.putHist("ARG");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[12].get(this.curWordIndex)){

if(!this.Type())

return false;

if(this.curWordIndex != 1) // идентификатор

return false;

this.curWordIndex = this.getWordIndex();

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

return this.ARG\_0();

}

return false;

},

FunctionBody: function(){

if(this.cntHist > 0)

this.putHist("FunctionBody");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[13].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Body())

return false;

if(this.curWordIndex != 45) // }

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Body: function(){

if(this.cntHist > 0)

this.putHist("Body");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[14].get(this.curWordIndex)){

if(!this.Code())

return false;

return this.Return();

}

return false;

},

Return: function(){

if(this.cntHist > 0)

this.putHist("Return");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[15].get(this.curWordIndex)){

{}

return true;

}

if((this.curWordIndex >= 0) && this.selSet[16].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Expression())

return false;

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Expression: function(){

if(this.cntHist > 0)

this.putHist("Expression");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[17].get(this.curWordIndex)){

if(!this.X())

return false;

return this.Y();

}

if((this.curWordIndex >= 0) && this.selSet[18].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.LgExp())

return false;

if(this.curWordIndex != 46) // ЛК

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

LogExpression: function(){

if(this.cntHist > 0)

this.putHist("LogExpression");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[19].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.LgExp())

return false;

if(this.curWordIndex != 46) // ЛК

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

AgExp: function(){

if(this.cntHist > 0)

this.putHist("AgExp");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[20].get(this.curWordIndex)){

if(!this.X())

return false;

return this.Y();

}

return false;

},

Y: function(){

if(this.cntHist > 0)

this.putHist("Y");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[21].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.AgExp();

}

if((this.curWordIndex >= 0) && this.selSet[22].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.AgExp();

}

if((this.curWordIndex >= 0) && this.selSet[23].get(this.curWordIndex)){

{}

return true;

}

return false;

},

X: function(){

if(this.cntHist > 0)

this.putHist("X");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[24].get(this.curWordIndex)){

if(!this.R())

return false;

return this.Z();

}

return false;

},

Z: function(){

if(this.cntHist > 0)

this.putHist("Z");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[25].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.X();

}

if((this.curWordIndex >= 0) && this.selSet[26].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.X();

}

if((this.curWordIndex >= 0) && this.selSet[27].get(this.curWordIndex)){

{}

return true;

}

return false;

},

R: function(){

if(this.cntHist > 0)

this.putHist("R");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[28].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.AgExp())

return false;

if(this.curWordIndex != 9) // )

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[29].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[30].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

LgExp: function(){

if(this.cntHist > 0)

this.putHist("LgExp");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[31].get(this.curWordIndex)){

if(!this.XL())

return false;

return this.YL();

}

return false;

},

YL: function(){

if(this.cntHist > 0)

this.putHist("YL");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[32].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.LgExp();

}

if((this.curWordIndex >= 0) && this.selSet[33].get(this.curWordIndex)){

{}

return true;

}

return false;

},

XL: function(){

if(this.cntHist > 0)

this.putHist("XL");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[34].get(this.curWordIndex)){

if(!this.RL())

return false;

return this.ZL();

}

return false;

},

ZL: function(){

if(this.cntHist > 0)

this.putHist("ZL");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[35].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.XL();

}

if((this.curWordIndex >= 0) && this.selSet[36].get(this.curWordIndex)){

{}

return true;

}

return false;

},

RL: function(){

if(this.cntHist > 0)

this.putHist("RL");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[37].get(this.curWordIndex)){

return this.AgExp();

}

if((this.curWordIndex >= 0) && this.selSet[38].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[39].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Code: function(){

if(this.cntHist > 0)

this.putHist("Code");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[40].get(this.curWordIndex)){

if(!this.Code\_0())

return false;

return this.Code\_1();

}

return false;

},

Block: function(){

if(this.cntHist > 0)

this.putHist("Block");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[41].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Block\_0())

return false;

if(this.curWordIndex != 27) // end

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Operator: function(){

if(this.cntHist > 0)

this.putHist("Operator");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[42].get(this.curWordIndex)){

return this.OperatorDeclaration();

}

if((this.curWordIndex >= 0) && this.selSet[43].get(this.curWordIndex)){

return this.LoopOperator();

}

if((this.curWordIndex >= 0) && this.selSet[44].get(this.curWordIndex)){

return this.ConditionalOperator();

}

if((this.curWordIndex >= 0) && this.selSet[45].get(this.curWordIndex)){

return this.SelectOperator();

}

if((this.curWordIndex >= 0) && this.selSet[46].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 6) // вывод

return false;

this.curWordIndex = this.getWordIndex();

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

if(!this.Expression())

return false;

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[47].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 7) // ввод

return false;

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 1) // идентификатор

return false;

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

OperatorDeclaration: function(){

if(this.cntHist > 0)

this.putHist("OperatorDeclaration");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[48].get(this.curWordIndex)){

if(!this.OperatorDeclaration\_0())

return false;

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

AOLeft: function(){

if(this.cntHist > 0)

this.putHist("AOLeft");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[49].get(this.curWordIndex)){

if(!this.AOLeft\_0())

return false;

if(!this.Var())

return false;

return this.AOLeft\_1();

}

return false;

},

Var: function(){

if(this.cntHist > 0)

this.putHist("Var");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[50].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.Var\_0();

}

return false;

},

AORight: function(){

if(this.cntHist > 0)

this.putHist("AORight");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[51].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Expression())

return false;

return this.AORight\_0();

}

return false;

},

LoopOperator: function(){

if(this.cntHist > 0)

this.putHist("LoopOperator");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[52].get(this.curWordIndex)){

if(!this.Loop())

return false;

return this.Until();

}

return false;

},

Loop: function(){

if(this.cntHist > 0)

this.putHist("Loop");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[53].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.OB();

}

return false;

},

OB: function(){

if(this.cntHist > 0)

this.putHist("OB");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[54].get(this.curWordIndex)){

return this.OperatorDeclaration();

}

if((this.curWordIndex >= 0) && this.selSet[55].get(this.curWordIndex)){

return this.LoopOperator();

}

if((this.curWordIndex >= 0) && this.selSet[56].get(this.curWordIndex)){

return this.ConditionalOperator();

}

if((this.curWordIndex >= 0) && this.selSet[57].get(this.curWordIndex)){

return this.SelectOperator();

}

if((this.curWordIndex >= 0) && this.selSet[58].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 6) // вывод

return false;

this.curWordIndex = this.getWordIndex();

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

if(!this.Expression())

return false;

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[59].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 7) // ввод

return false;

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 1) // идентификатор

return false;

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[60].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(!this.Block\_0())

return false;

if(this.curWordIndex != 27) // end

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Until: function(){

if(this.cntHist > 0)

this.putHist("Until");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[61].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.LogExpression();

}

return false;

},

ConditionalOperator: function(){

if(this.cntHist > 0)

this.putHist("ConditionalOperator");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[62].get(this.curWordIndex)){

if(!this.When())

return false;

if(!this.Then())

return false;

if(!this.Other())

return false;

if(this.curWordIndex != 42) // wend

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

When: function(){

if(this.cntHist > 0)

this.putHist("When");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[63].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.LogExpression();

}

return false;

},

Then: function(){

if(this.cntHist > 0)

this.putHist("Then");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[64].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.OB();

}

return false;

},

Other: function(){

if(this.cntHist > 0)

this.putHist("Other");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[65].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.OB();

}

if((this.curWordIndex >= 0) && this.selSet[66].get(this.curWordIndex)){

{}

return true;

}

return false;

},

SelectOperator: function(){

if(this.cntHist > 0)

this.putHist("SelectOperator");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[67].get(this.curWordIndex)){

if(!this.Select())

return false;

if(!this.SelectOperator\_0())

return false;

if(this.curWordIndex != 38) // send

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Select: function(){

if(this.cntHist > 0)

this.putHist("Select");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[68].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return this.Expression();

}

return false;

},

Case: function(){

if(this.cntHist > 0)

this.putHist("Case");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[69].get(this.curWordIndex)){

if(!this.CaseHead())

return false;

return this.CaseBody();

}

return false;

},

CaseHead: function(){

if(this.cntHist > 0)

this.putHist("CaseHead");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[70].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 8) // (

return false;

this.curWordIndex = this.getWordIndex();

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

if(!this.Const())

return false;

if(this.curWordIndex != 9) // )

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Const: function(){

if(this.cntHist > 0)

this.putHist("Const");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[71].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[72].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

CaseBody: function(){

if(this.cntHist > 0)

this.putHist("CaseBody");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[73].get(this.curWordIndex)){

if(!this.OB())

return false;

return this.CaseBody\_0();

}

return false;

},

FileHeader: function(){

if(this.cntHist > 0)

this.putHist("FileHeader");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[74].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 1) // идентификатор

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

return false;

},

Program\_0: function(){

if(this.cntHist > 0)

this.putHist("Program\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[75].get(this.curWordIndex)){

if(!this.DataOrFunction())

return false;

return this.Program\_1();

}

return false;

},

Program\_1: function(){

if(this.cntHist > 0)

this.putHist("Program\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[76].get(this.curWordIndex)){

if(!this.DataOrFunction())

return false;

return this.Program\_1();

}

if((this.curWordIndex >= 0) && this.selSet[77].get(this.curWordIndex)){

{}

return true;

}

return false;

},

Arguments\_0: function(){

if(this.cntHist > 0)

this.putHist("Arguments\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[78].get(this.curWordIndex)){

if(!this.ARG())

return false;

return this.Arguments\_0();

}

if((this.curWordIndex >= 0) && this.selSet[79].get(this.curWordIndex)){

{}

return true;

}

return false;

},

ARG\_0: function(){

if(this.cntHist > 0)

this.putHist("ARG\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[80].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[81].get(this.curWordIndex)){

{}

return true;

}

return false;

},

Code\_0: function(){

if(this.cntHist > 0)

this.putHist("Code\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[82].get(this.curWordIndex)){

if(!this.Operator())

return false;

return this.Code\_0();

}

if((this.curWordIndex >= 0) && this.selSet[83].get(this.curWordIndex)){

{}

return true;

}

return false;

},

Code\_1: function(){

if(this.cntHist > 0)

this.putHist("Code\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[84].get(this.curWordIndex)){

if(!this.Block())

return false;

return this.Code\_1();

}

if((this.curWordIndex >= 0) && this.selSet[85].get(this.curWordIndex)){

{}

return true;

}

return false;

},

Block\_0: function(){

if(this.cntHist > 0)

this.putHist("Block\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[86].get(this.curWordIndex)){

if(!this.Operator())

return false;

return this.Block\_1();

}

return false;

},

Block\_1: function(){

if(this.cntHist > 0)

this.putHist("Block\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[87].get(this.curWordIndex)){

if(!this.Operator())

return false;

return this.Block\_1();

}

if((this.curWordIndex >= 0) && this.selSet[88].get(this.curWordIndex)){

{}

return true;

}

return false;

},

OperatorDeclaration\_0: function(){

if(this.cntHist > 0)

this.putHist("OperatorDeclaration\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[89].get(this.curWordIndex)){

if(!this.AOLeft())

return false;

if(!this.OperatorDeclaration\_2())

return false;

return this.OperatorDeclaration\_1();

}

return false;

},

OperatorDeclaration\_1: function(){

if(this.cntHist > 0)

this.putHist("OperatorDeclaration\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[90].get(this.curWordIndex)){

if(!this.AOLeft())

return false;

if(!this.OperatorDeclaration\_3())

return false;

return this.OperatorDeclaration\_1();

}

if((this.curWordIndex >= 0) && this.selSet[91].get(this.curWordIndex)){

{}

return true;

}

return false;

},

AOLeft\_0: function(){

if(this.cntHist > 0)

this.putHist("AOLeft\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[92].get(this.curWordIndex)){

return this.Type();

}

if((this.curWordIndex >= 0) && this.selSet[93].get(this.curWordIndex)){

{}

return true;

}

return false;

},

AOLeft\_1: function(){

if(this.cntHist > 0)

this.putHist("AOLeft\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[94].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[95].get(this.curWordIndex)){

{}

return true;

}

return false;

},

Var\_0: function(){

if(this.cntHist > 0)

this.putHist("Var\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[96].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 2) // константа

return false;

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 20) // ]

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[97].get(this.curWordIndex)){

{}

return true;

}

return false;

},

AORight\_0: function(){

if(this.cntHist > 0)

this.putHist("AORight\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[98].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[99].get(this.curWordIndex)){

{}

return true;

}

return false;

},

SelectOperator\_0: function(){

if(this.cntHist > 0)

this.putHist("SelectOperator\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[100].get(this.curWordIndex)){

if(!this.Case())

return false;

return this.SelectOperator\_1();

}

return false;

},

SelectOperator\_1: function(){

if(this.cntHist > 0)

this.putHist("SelectOperator\_1");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[101].get(this.curWordIndex)){

if(!this.Case())

return false;

return this.SelectOperator\_1();

}

if((this.curWordIndex >= 0) && this.selSet[102].get(this.curWordIndex)){

{}

return true;

}

return false;

},

CaseBody\_0: function(){

if(this.cntHist > 0)

this.putHist("CaseBody\_0");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[103].get(this.curWordIndex)){

this.curWordIndex = this.getWordIndex();

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

if(this.curWordIndex != 16) // ;

return false;

this.curWordIndex = this.getWordIndex();

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

return true;

}

if((this.curWordIndex >= 0) && this.selSet[104].get(this.curWordIndex)){

{}

return true;

}

return false;

},

OperatorDeclaration\_2: function(){

if(this.cntHist > 0)

this.putHist("OperatorDeclaration\_2");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[105].get(this.curWordIndex)){

return this.AORight();

}

if((this.curWordIndex >= 0) && this.selSet[106].get(this.curWordIndex)){

{}

return true;

}

return false;

},

OperatorDeclaration\_3: function(){

if(this.cntHist > 0)

this.putHist("OperatorDeclaration\_3");

this.cCnt += 1;

if((this.curWordIndex >= 0) && this.selSet[107].get(this.curWordIndex)){

return this.AORight();

}

if((this.curWordIndex >= 0) && this.selSet[108].get(this.curWordIndex)){

{}

return true;

}

return false;

},

\_Program\_:0

};

(function(){

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

document.getElementById("syntName").innerText = " рекурсивный спуск, процедурная реализация.";

})();