**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ имени Н.Э.БАУМАНА  
(национальный исследовательский университет)»**

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа № 7.1**

“Синтаксический анализатор на основе предсказывающего анализа”

по дисциплине «Конструирование компиляторов»

Вариант 1 (JAVA)

Работу выполнил

студент группы ИУ9-62Б

Сербин Денис

Москва, 2022

# **1. Цель работы**

# Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

**2. Исходные данные**

В данной лабораторной работе требуется разработать синтаксический анализатор на основе предсказывающего анализа, который принимая на входе текст на входном языке, на выходе порождает дерево вывода для входного текста. В качестве входного языка должен выступать язык представления правил грамматики, варианты лексики и синтаксиса которого можно восстановить по примерам.

# **3. Задание**

Выполнение лабораторной работы состоит из следующих этапов:

1. Составление описаний лексической структуры и грамматики входного языка на основе примера.

2. Разработка лексического анализатора для входного языка. Лексический анализ может выполняться любым из ранее изученных способов.

3. Составление (вручную) таблицы предсказывающего разбора для входного языка.

4. Разработка алгоритма предсказывающего разбора, работающего на основе порождённой таблицы. Реализация этого алгоритма.

В отчёте по лабораторной работе обязательно должно присутствовать описание синтаксиса входного языка - как неформальное (своими словами), так и формализованное (лексическая структура и грамматика).

**4. Индивидуальный вариант**

# объявления

non-terminal E, E1, T, T1, F;

terminal '+', '\*', '(',')', n;

# правила грамматики

E ::= TE1;

E1 ::= ‘+' T E1 | epsilon;

T ::= F T1;

T1 ::= '\*' F T1 | epsilon;

F ::= n | '(' E ')';

axiom E;

# **5. Результат выполнения**

Cинтаксис входного языка:

S ::= DEF R\_LST AXIOM  
DEF ::= D\_N D\_T  
D\_N ::= "non-terminal" N N\_LST ';'  
N\_LST ::= ',' N N\_LST | eps  
D\_T ::= "terminal" T T\_LST ';'  
T\_LST ::= ',' T T\_LST | eps  
R\_LST ::= R R\_LST | eps  
R ::= N "::=" E E\_LST ';'  
E\_LST ::= '|' E E\_LST | eps  
E ::= SYM SYM\_LST | "epsilon"  
SYM\_LST ::= SYM SYM\_LST | eps  
SYM ::= N | T  
AXIOM ::= "axiom" N N\_LST ';'  
N ::= "[A-Z]+1?"  
T ::= "\'(\*|+|(|))\'|[a-z]"

S – стартовый нетерминал, который переходит в DEF- нетерминал объявлений терминалов и нетерминалов, R\_LST – список правил, AXIOM – объявление аксиомы. DEF переходит в D\_N и D\_T – объявление нетерминалов и терминалов соответственно. D\_N включает в себя слово “non-terminal”, N – нетерминал и N\_LST – список нетерминалов и знак ‘;’. N\_LST состоит из знака ‘,’, N и N\_LST, либо N\_LST переходит в epsilon. D\_T определяется аналогично D\_N. R\_LST может перейти в R – определенное правило и R\_LST, либо в epsilon. R состоит из нетерминала N, знака “::=”, E – выражение и E\_LST – списка выражений и знака ‘;’. E\_LST переходи либо в E и E\_LST, либо в epsilon. E состоит из SYM- символа и SYM\_LST – список символов, либо из слова “epsilon”. SYM\_LST определяется аналогично R\_LST. SYM может перейти либо в T, либо в N. AXIOM состоит из слова “axiom” и нетерминалов N, N\_LST и знака ‘;’.

Таблицы FIRST и FOLLOW:

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| S | «non-terminal» | $ |
| DEF | «non-terminal» | N, eps, «axiom», $ |
| D\_N | «non-terminal» | «terminal», N, eps, «axiom», $ |
| N\_LST | ',', eps | ';' |
| D\_T | «terminal» | N, eps, «axiom», $ |
| T\_LST | ',', eps | ';' |
| R\_LST | N, eps | «axiom», $ |
| R | N | N, eps, «axiom», $ |
| E\_LST | '|', eps | ';' |
| E | N, T, «epsilon» | '|', eps, ';' |
| SYM\_LST | N, T, eps | '|', eps, ';' |
| SYM | N, T | N, T, eps, '|', ';' |
| AXIOM | «axiom» | $ |

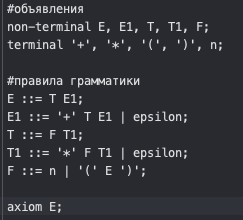
Таблица предсказывающего разбора для входного языка:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | «non-terminal» | «terminal» | «axiom» | «epsilon» | ',' | '|' | ';' | ::= | N | T | $ |
| S | DEF R\_LST AXIOM |  |  |  |  |  |  |  |  |  |  |
| DEF | D\_N D\_T |  |  |  |  |  |  |  |  |  |  |
| D\_N | "non-terminal" N N\_LST ';' |  |  |  |  |  |  |  |  |  |  |
| N\_LST |  |  |  |  | ',' N N\_LST |  | eps |  |  |  |  |
| D\_T |  | "terminal" T T\_LST ';' |  |  |  |  |  |  |  |  |  |
| T\_LST |  |  |  |  | ',' T T\_LST |  | eps |  |  |  |  |
| R\_LST |  |  | eps |  |  |  |  |  | R R\_LST |  | eps |
| R |  |  |  |  |  |  |  |  | N "::=" E E\_LST ';' |  |  |
| E\_LST |  |  |  |  |  | '|' E E\_LST | eps |  |  |  |  |
| E |  |  |  | «epsilon» |  |  |  |  | SYM SYM\_LST | SYM SYM\_LST |  |
| SYM\_LST |  |  |  |  |  | eps | eps |  | SYM SYM\_LST | SYM SYM\_LST |  |
| SYM |  |  |  |  |  |  |  |  | N | T |  |
| AXIOM |  |  | "axiom" N ';' |  |  |  |  |  |  |  |  |

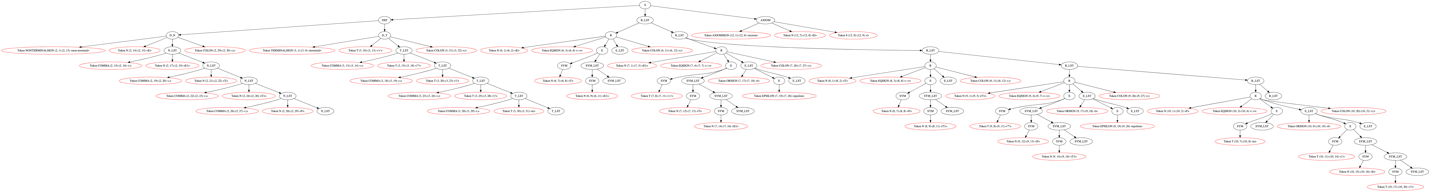
Код грамматики и таблицы предсказывающего анализа:

**private static** HashMap<String, Rules> getGrammar() {  
 HashMap<String, Rules> rules = **new** HashMap<>();  
 rules.put("S", **new** Rules(**new** RHS(  
 **new** Nonterm("DEF"),  
 **new** Nonterm("R\_LST"),  
 **new** Nonterm("AXIOM")  
 )));  
 rules.put("DEF",  
 **new** Rules(**new** RHS(  
 **new** Nonterm("D\_N"),  
 **new** Nonterm("D\_T")  
 )));  
 rules.put("D\_N", **new** Rules(**new** RHS(  
 **new** Term("NONTERMINALSIGN"),  
 **new** Term("N"),  
 **new** Nonterm("N\_LST"),  
 **new** Term("COLON")  
 )));  
 rules.put("N\_LST", **new** Rules(  
 **new** RHS(  
 **new** Term("COMMA"),  
 **new** Term("N"),  
 **new** Nonterm("N\_LST")  
 ),  
 **new** Epsilon()  
 ));  
 rules.put("D\_T", **new** Rules(**new** RHS(  
 **new** Term("TERMINALSIGN"),  
 **new** Term("T"),  
 **new** Nonterm("T\_LST"),  
 **new** Term("COLON")  
 )));  
 rules.put("T\_LST", **new** Rules(  
 **new** RHS(  
 **new** Term("COMMA"),  
 **new** Term("T"),  
 **new** Nonterm("T\_LST")  
 ),  
 **new** Epsilon()  
 ));  
 rules.put("R\_LST",**new** Rules(  
 **new** RHS(  
 **new** Nonterm("R"),  
 **new** Nonterm("R\_LST")  
 ),  
 **new** Epsilon()  
 ));  
 rules.put("R", **new** Rules(**new** RHS(  
 **new** Term("N"),  
 **new** Term("EQSIGN"),  
 **new** Nonterm("E"),  
 **new** Nonterm("E\_LST"),  
 **new** Term("COLON")  
 )));  
 rules.put("E\_LST", **new** Rules(  
 **new** RHS(  
 **new** Term("ORSIGN"),  
 **new** Nonterm("E"),  
 **new** Nonterm("E\_LST")  
 ),  
 **new** Epsilon()  
 ));  
 rules.put("E", **new** Rules(  
 **new** RHS(  
 **new** Nonterm("SYM"),  
 **new** Nonterm("SYM\_LST")  
 ),  
 **new** RHS(  
 **new** Term("EPSILON")  
 )  
 ));  
 rules.put("SYM\_LST", **new** Rules(  
 **new** RHS(  
 **new** Nonterm("SYM"),  
 **new** Nonterm("SYM\_LST")  
 ),  
 **new** Epsilon()  
 ));  
 rules.put("SYM", **new** Rules(  
 **new** RHS(  
 **new** Term("N")  
 ),  
 **new** RHS(  
 **new** Term("T")  
 )  
 ));  
 rules.put("AXIOM", **new** Rules(  
 **new** RHS(  
 **new** Term("AXIOMSIGN"),  
 **new** Term("N"),  
 **new** Term("COLON")  
 )  
 ));  
  
 **return** rules;  
 }  
  
 **private static** RHS[][] analiseTable() {  
 ArrayList<String> T = *termList*;  
 ArrayList<String> N = *nontermList*;  
 HashMap<String, Rules> rules = *grammarList*;  
 **int** m = N.size();  
 **int** n = T.size();  
 RHS[][] q = **new** RHS[m][n];  
 **for** (RHS[] line: q) {  
 Arrays.*fill*(line, **new** Error());  
 }  
  
 q[N.indexOf("S")][T.indexOf("NONTERMINALSIGN")] = rules.get("S").get(0);  
  
 q[N.indexOf("DEF")][T.indexOf("NONTERMINALSIGN")] = rules.get("DEF").get(0);  
  
 q[N.indexOf("D\_N")][T.indexOf("NONTERMINALSIGN")] = rules.get("D\_N").get(0);  
  
 q[N.indexOf("N\_LST")][T.indexOf("COMMA")] = rules.get("N\_LST").get(0);  
 q[N.indexOf("N\_LST")][T.indexOf("COLON")] = rules.get("N\_LST").get(1);  
  
 q[N.indexOf("D\_T")][T.indexOf("TERMINALSIGN")] = rules.get("D\_T").get(0);  
  
 q[N.indexOf("T\_LST")][T.indexOf("COMMA")] = rules.get("T\_LST").get(0);  
 q[N.indexOf("T\_LST")][T.indexOf("COLON")] = rules.get("T\_LST").get(1);  
  
 q[N.indexOf("R\_LST")][T.indexOf("AXIOMSIGN")] = rules.get("R\_LST").get(1);  
 q[N.indexOf("R\_LST")][T.indexOf("N")] = rules.get("R\_LST").get(0);  
 q[N.indexOf("R\_LST")][T.indexOf(Term.*EOF*)] = rules.get("R\_LST").get(1);  
  
 q[N.indexOf("R")][T.indexOf("N")] = rules.get("R").get(0);  
  
 q[N.indexOf("E\_LST")][T.indexOf("ORSIGN")] = rules.get("E\_LST").get(0);  
 q[N.indexOf("E\_LST")][T.indexOf("COLON")] = rules.get("E\_LST").get(1);  
  
 q[N.indexOf("E")][T.indexOf("EPSILON")] = rules.get("E").get(1);  
 q[N.indexOf("E")][T.indexOf("N")] = rules.get("E").get(0);  
 q[N.indexOf("E")][T.indexOf("T")] = rules.get("E").get(0);  
  
 q[N.indexOf("SYM\_LST")][T.indexOf("ORSIGN")] = rules.get("SYM\_LST").get(1);  
 q[N.indexOf("SYM\_LST")][T.indexOf("COLON")] = rules.get("SYM\_LST").get(1);  
 q[N.indexOf("SYM\_LST")][T.indexOf("N")] = rules.get("SYM\_LST").get(0);  
 q[N.indexOf("SYM\_LST")][T.indexOf("T")] = rules.get("SYM\_LST").get(0);  
  
 q[N.indexOf("SYM")][T.indexOf("N")] = rules.get("SYM").get(0);  
 q[N.indexOf("SYM")][T.indexOf("T")] = rules.get("SYM").get(1);  
  
 q[N.indexOf("AXIOM")][T.indexOf("AXIOMSIGN")] = rules.get("AXIOM").get(0);  
   
 **return** q;  
 }

Входной файл для программы:



Результат вывода программы реализован в виде файла формата dot:

**6. Вывод**

В рамках данной лабораторной работы был изучен алгоритм построения таблиц предсказывающего анализатора