UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
COMP263 - Compiladores

LINGUAGEM DUMA: ESPECIFICAÇÃO DOS TOKENS

Eduarda Tatiane Caetano Chagas
Marcos Gleysson Silva do Nascimento

Maceió-AL
2017.1

Sumário

CAPÍTULO 1

ESPECIFICAÇÃO DOS TOKENS

1.1 Especificação da linguagem de programação

A linguagem de programação que será utilizada para implementação do analisador léxico e sintático da Linguagem DUMA será **Java**. A escolha dessa linguagem foi feita baseada nos recursos e ferramentas que a mesma possui e oferece na área de Compiladores para possibilitar a construção de analisadores léxicos e sintáticos.

1.2 Enumeração com as categorias dos *tokens*

Os *tokens* da linguagem DUMA serão classificados segundo a enumeração de categorias abaixo. Note que foi utilizado a sintaxe da linguagem de programação Java que se trata da linguagem escolhida para implementação dos analisadores.

```
public enum TokenCategory{

    prDuma, prInitium, prConst, prVar, prFun,
    id, tdInanis, tdInt, tdReal, tdLit, tdBool,
    tdSermo, prMatrix, escBegin, escEnd, paramBegin,
    paramEnd, comment, sepVirg, termCmd, cteNumInt,
    cteNumReal, cteLit, cteSermo, cteBool, selSi,
    selAliud, selSialiud, repQuia, repDum, repFacite,
    repSpatium, repIn, opAritAd, opAritMul, opLogAnd,
    opLogOr, opLogNeg, opAritUn, opRel1, opRel2,
    opCon, vetBegin, vetEnd

}
```

1.3 Especificação dos *tokens* da linguagem

1.3.1 Expressões regulares auxiliares

letter = ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’ | ’g’ | ’h’ | ’i’ | ’j’ | ’k’ | ’l’ | ’m’ | ’n’ | ’o’ | ’p’ | ’q’ | ’r’ | ’s’ | ’t’ | ’u’ | ’v’ | ’w’ | ’x’ | ’y’ | ’z’ | ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’ | ’G’ | ’H’ | ’I’ | ’J’ | ’K’ | ’L’ | ’M’ | ’N’ | ’O’ | ’P’ | ’Q’ | ’R’ | ’S’ | ’T’ | ’U’ | ’V’ | ’W’ | ’X’ | ’Y’ | ’Z’ ;

digit = ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’;

symbol = ’ ’ | ’.’ | ’,’ | ’:’ | ’;’ | ’?’ | ’!’ | ’+’ | ’-’ | ’∗’ | ’\’ | ’/’ | ’_’ | ’%’ | ’@’ | ’&’ | ’#’ | ’$’ | ’<’ | ’>’ | ’=’ | ’(’ | ’)’ | ’[’ | ’]’ | ’{’ | ’}’ | ’|’ | ’ ’ ’ | ’ "’ ;

1.3.2 Lexemas

Identificador:

id = (’letter’ | ’_’)(’letter’ | ’_’ | ’digit’)* ;

Tipos de dados primitivos:

tdInanis = ’inanis’;
tdInt = ’integer’;
tdReal = ’real’;
tdLit = ’litterae’;
tdBool = ’boolean’;
tdSermo = ’sermo’;

Outras palavras reservadas:

prInitium = ’initium’;
prDuma = ’duma’;
prMatrix = ’matrix’;
prConst = ’const’;
prVar = ’var’;
prFun = ’fun’;

Delimitadores:

1. Escopo:
   escBegin = ’{’;
   escEnd = ’}’
2. Parâmetros:
   paramBegin = ’(’;
   paramEnd = ’)’;
3. Arrays:
   vetBegin = ’[’;
   vetEnd = ’]’;
4. Comentários:
   comment = ’#’;
5. Terminador de comando:
   termCmd = ’;’ ;
6. Separadores:
   sepVirg = ’,’ ;

Constantes de tipos:

cteNumInt = (’digit’)(’digit’)* ;

cteNumReal = (’digit’)(’digit’)*(’.’)(’digit’)(’digit’)* ;

cteBool = (’true’ | ’false’) ;

cteLit = (”’)(’letter’ | ’digit’ | ’symbol’)(”’) ;

cteSermo = (’”’)(’letter’ | ’digit’ | ’symbol’)*(’”’) ;

Comandos de Seleção:

selSi = ’si’;

selAliud = ’aliud’;

selSialiud = ’sialiud’;

Comandos de Repetição:

repQuia = ’quia’;

repFacite = ’facite’;

repDum = ’dum’;

repSpatium = ’spatium’;

repIn = ’in’;

Operadores Aritméticos:

opAritAd = ’+’ | ’-’;

opAritMul = ’*’ | ’ /’ | ’%’;

Operadores Lógicos:

opLogNeg = ’!’;

opLogAnd = ’&&’;

opLogOr = ’||’;

Operadores Relacionais:

opRel1 = ’>’ | ’<’ | ’>=’ | ’<=’;

opRel2 = ’==’ | ’!=’;

Operador de concatenação:

opCon = ’.’;

Operador Unário:

opAriUn = ’-’;

Observação: Será levado em consideração um tratamento especial para o tratamento do operador aritmético unário (-) e do operador aritmético binário de mesmo símbolo. Como estratégia, analisaremos os casos em que este operador se configura como operador binário, pois, desse modo, existem poucas possibilidades de verificação, já que o operador será tido como binário apenas se estiver entre dois operandos (que pode ser indicado por identificador, constante numérica ou parênteses).