UFSC - Universidade Federal de Santa Catarina
INE - Departamento de Informática e Estatística

INE5426 - Construção de Compiladores Analisador Léxico e Analisador Sintático

08/10/2019

Lucas Henrique Gonçalves Wodtke (16202258)
e
Marcelo Emilio Vendramin (16200653)

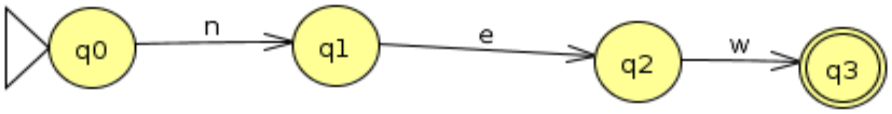
Tarefa AL

1. **identificação dos tokens:**

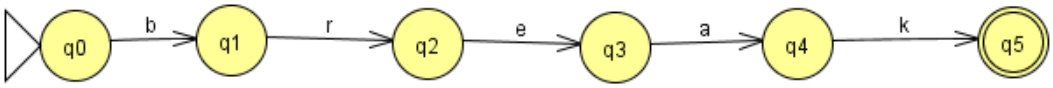
'new' = 0
'break' = 1
'ident' = 2
'int_constant' = 3
'int' = 4
'float' = 5
'string' = 6
'float_constant' = 7
'print' = 8
'read' = 9
'return' = 10
'for' = 11
'if' = 12
')' = 13
'else' = 14
'null' = 15
'string_constant' = 16
'{' = 17
'}' = 18
';' = 19
'[' = 20
']' = 21
'<' = 22
'>' = 23
'<=' = 24
'>=' = 25
'==' = 26
'!=' = 27
'+' = 28
'-' = 29

'=' = 30
'(' = 31
'*' = 32
'/' = 33
'%' = 34
'else' = 35

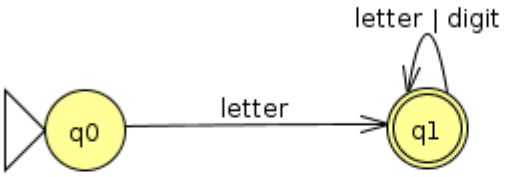
2. **construção dos diagramas de transição para cada token:**
   Token 'new'



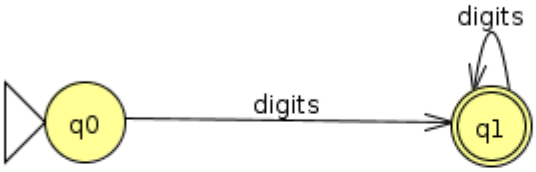
Token 'break'



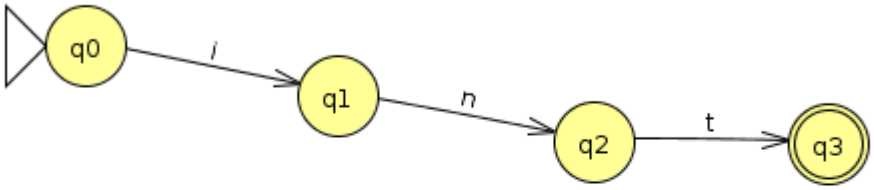
Token 'ident'



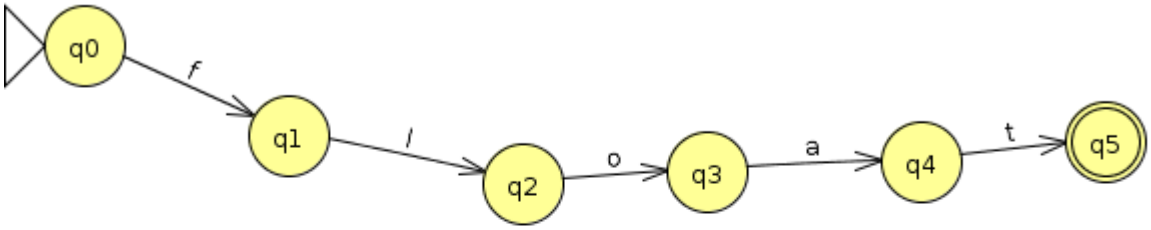
Token 'int_constant'



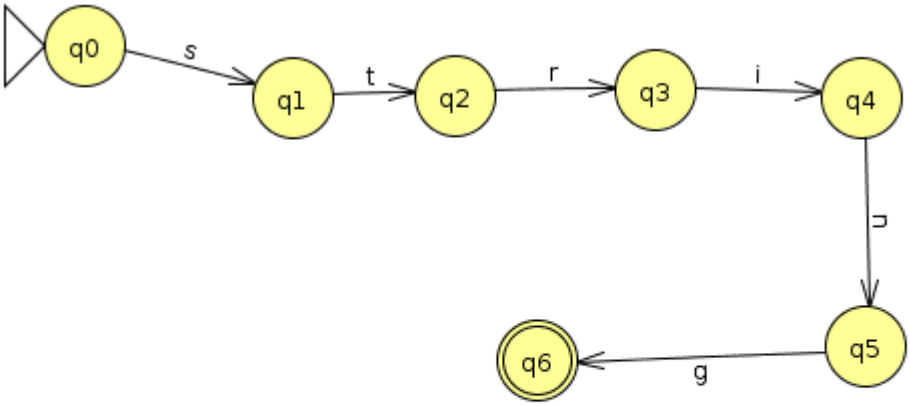
Token 'int'



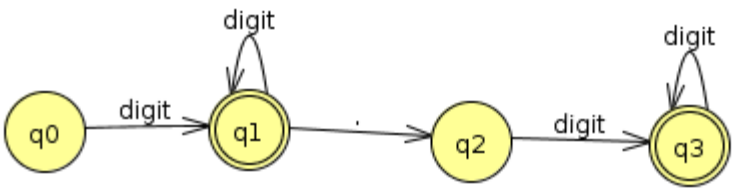
Token 'float'



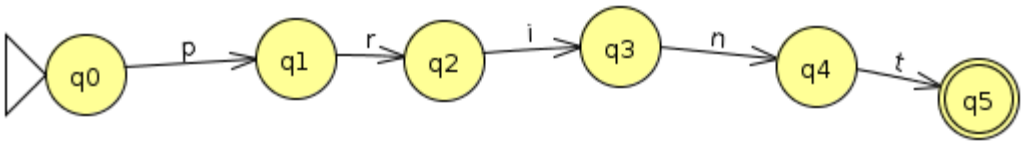
Token 'string'



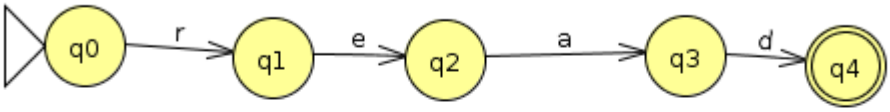
Token 'float_constant'



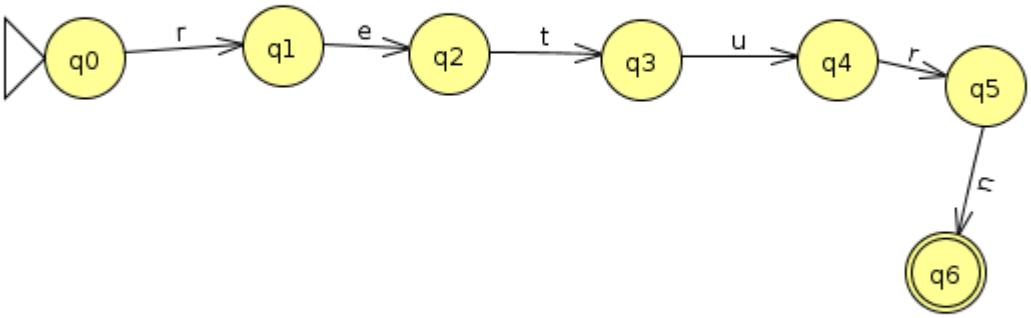
Token 'print'



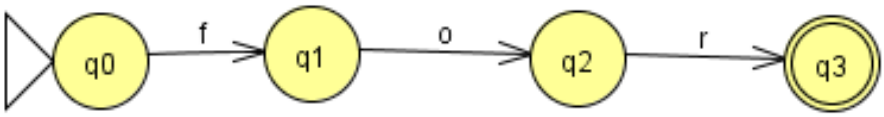
Token 'read'



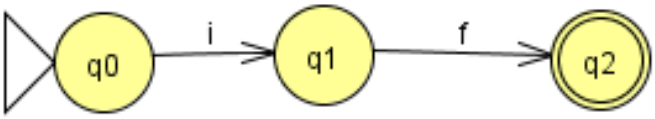
Token 'return'



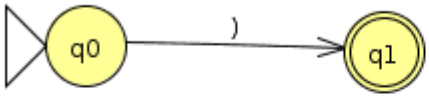
Token 'for'



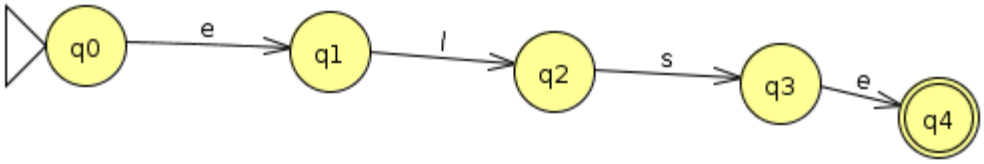
Token 'if'



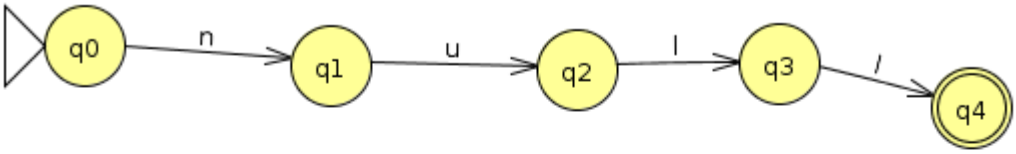
Token ')'



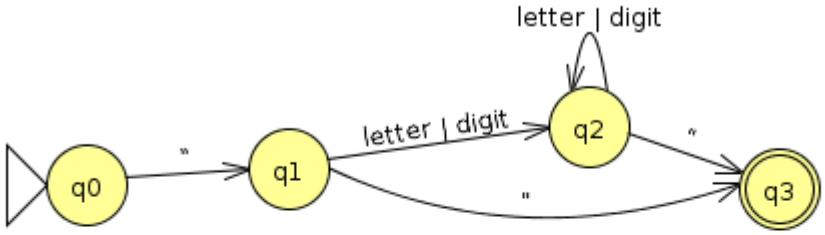
Token 'else'



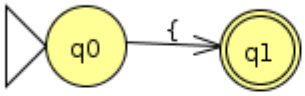
Token 'null'



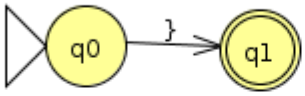
Token 'string_constant'



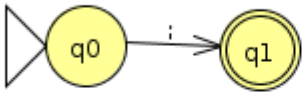
Token '{'



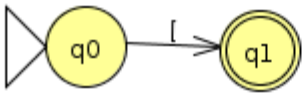
Token '}'



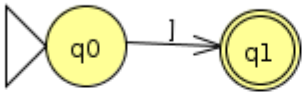
Token ';'



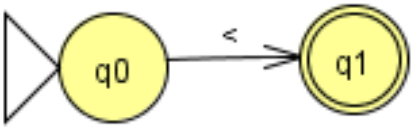
Token '['



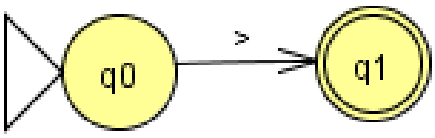
Token ']'



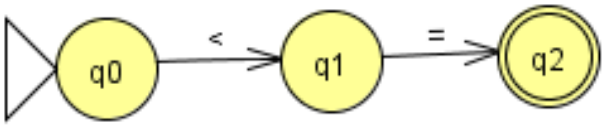
Token '<'



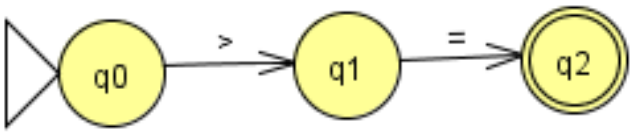
Token '>'



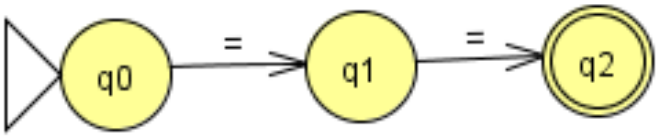
Token '<='



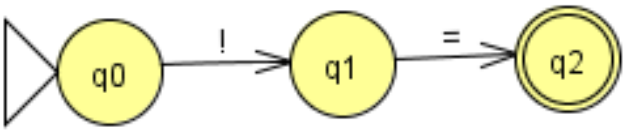
Token '>='



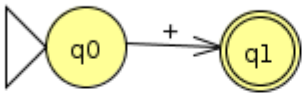
Token '=='



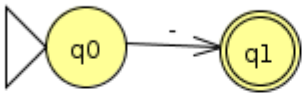
Token '!='



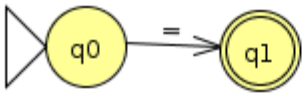
Token '+'



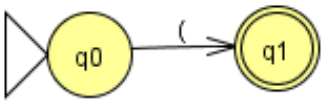
Token '-'



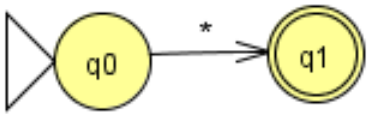
Token '='



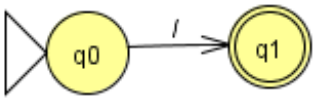
Token '('



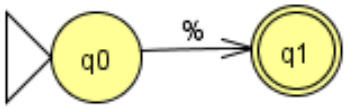
Token '*'



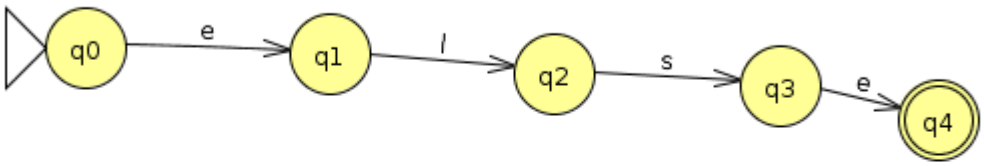
Token '/'



Token '%'



Token 'else'



3. **Se não usou ferramenta, uma descrição da implementação do analisador léxico (Usou diagramas de transição? Quais? Quantos? Se não usou diagramas de transição, então o que foi usado?):**

A implementação do analisador léxico para a gramática CC-2019-2, foi feita na linguagem Java,sem o uso de ferramentas que constrõem um analisador léxico.
As principais classes da implementação:
1.Token –Onde é definido a função de criação do objeto e seus atributos :

- String ‘valor’ representa a palavra extraida.
- Int token_id representa o id do token,um número inteiro único que o identifica.
- Int token_type representa qual dos tipos tokens ele é por exemplo: else,for,int_constant...
- Int token_line representa a linha em que o token está localizado.
- Int token_collum representa a coluna em que o token está localizado.

2.TabelaDeSimbolos – A Tabela de Simbolos foi representada através dessa classe,usando a estrutra Hash.
3. Tools- Nessa classe está contido o seguinte método mais relevante para Análise Léxica:

- Token[] analise_lexica(String string, TabelaDeSimbolos tab) aqui ocorre a extração e classificação dos tokens (criação dos objetos tokens),adição a tabela de simbolos se for necessário e a identificação de erros.
  Baseado na união de todos os diagramas de máquina de estados de cada token,com execeção dos que são palavras reservadas(o tratamento se dá com uma comparação direta),ao decorrer da leitura da entrada em um laço for,eles são comparados em uma estrutura de if´s que se assemelha no funcionamento de uma máquina de estados.

Etapa AS

1. CC-2019-2 está na forma BNF. Coloque-a na forma convencional de gramática. Chame tal gramática de CCC-2019-2.

Para a conversão para a forma convencional foi necessário a criação de novas Produções.

Program -> STATEMENT | ϵ

STATEMENT -> VARDECL ; | ATRIBSTAT ; | PRINTSTAT ; | READSTAT ; | RETURNSTAT ; | IFSTAT ; | FORSTAT ; | { STATELIST } | break ; | ;

VARDECL -> int ident | int ident VARDECLLOOP | float ident | float ident VARDECLLOOP | string ident | string ident VARDECLLOOP

VARDECLLOOP -> [int_constant] VARDECLLOOP | ϵ

ATRIBSTAT -> LVALUE = EXPRESSION | LVALUE = ALLOCEXPRESSION

PRINTSTAT -> print EXPRESSION

READSTAT -> read LVALUE

RETURNSTAT -> return

IFSTAT -> if (EXPRESSION) STATEMENT else STATEMENT | if (EXPRESSION) STATEMENT

FORSTAT -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT

STATELIST -> STATEMENT STATELIST | STATEMENT

ALLOCEXPRESSION -> new int EXPRESSIONLOOP | new float EXPRESSIONLOOP | new string EXPRESSIONLOOP

EXPRESSIONLOOP -> [EXPRESSION] EXPRESSIONLOOP | [EXPRESSION]

EXPRESSION -> NUMEXPRESSION | NUMEXPRESSION < NUMEXPRESSION | NUMEXPRESSION > NUMEXPRESSION | NUMEXPRESSION <= NUMEXPRESSION | NUMEXPRESSION >= NUMEXPRESSION | NUMEXPRESSION == NUMEXPRESSION | NUMEXPRESSION != NUMEXPRESSION

NUMEXPRESSION -> TERM TERMLOOP

TERMLOOP -> + TERM TERMLOOP | - TERM TERMLOOP | ϵ

TERM -> UNARYEXPR UNARYEXPRLOOP

UNARYEXPRLOOP -> * UNARYEXPR UNARYEXPRLOOP | / UNARYEXPR UNARYEXPRLOOP | % UNARYEXPR UNARYEXPRLOOP | ϵ

UNARYEXPR -> + FACTOR | - FACTOR | FACTOR

FACTOR -> int_constant | float_constant | string_constant | null | LVALUE | (EXPRESSION)
LVALUE -> ident LVALUELOOP

LVALUELOOP -> [EXPRESSION] LVALUELOOP | ϵ

2. CCC-2019-2 possui recursão á esquerda? Justifique detalhadamente sua resposta. Se ela tiver recursão á esquerda, então remova tal recursão.

Ela não apresenta recursão à esquerda .

Justificativa: Uma produção de uma gramática é considerada recursiva à esquerda se ela, de maneira direta ou indireta, deriva a si mesma por uma derivação mais à esquerda.

Com esta definição, foram analisadas todas as produções da gramática, constatando assim que nenhuma de suas derivações resultam em uma recursão à esquerda.

3. CCC-2019-2 está fatorada á esquerda? Justifique detalhadamente sua resposta. Se ela não tiver fatorada `a esquerda, então fatore.

Ela não está fatorada.

Uma gramática, para estar fatorada não deve possuir produções que contenham não-determinismo, ou seja, não podem existir produções que dada uma derivação mais à esquerda, resultem em mais de um caminho.Gerando a indecisão sobre qual procução aplicar quando duas ou mais produções iniciam com a mesma forma sentencial(direta ou indiretamente).

Program -> STATEMENT
| ϵ
STATEMENT -> VARDECL ;
| ATRIBSTAT ;
| PRINTSTAT ;
| READSTAT ;
| RETURNSTAT ;
| IFSTAT ;
| FORSTAT ;
| { STATELIST }
| break ;
| ;

VARDECL -> string ident VARDECL'
| float ident VARDECL'
| int ident VARDECL'

VARDECLLOOP -> [int_constant] VARDECLLOOP
| ϵ

ATRIBSTAT -> LVALUE = ATRIBSTAT'

PRINTSTAT -> print EXPRESSION

READSTAT -> read LVALUE

RETURNSTAT -> return

IFSTAT -> if (EXPRESSION) STATEMENT IFSTAT'

FORSTAT -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT

STATELIST -> STATEMENT STATELIST'

ALLOCEXPRESSION -> new ALLOCEXPRESSION'

EXPRESSIONLOOP -> [EXPRESSION] EXPRESSIONLOOP'

EXPRESSION -> NUMEXPRESSION EXPRESSION'

NUMEXPRESSION -> TERM TERMLOOP

TERMLOOP -> + TERM TERMLOOP
| - TERM TERMLOOP
| ϵ

TERM -> UNARYEXPR UNARYEXPRLOOP

UNARYEXPRLOOP -> * UNARYEXPR UNARYEXPRLOOP
| / UNARYEXPR UNARYEXPRLOOP
| % UNARYEXPR UNARYEXPRLOOP
| ϵ

UNARYEXPR -> + FACTOR
| - FACTOR
| FACTOR

FACTOR -> int_constant
| float_constant
| string_constant
| null
| LVALUE
| (EXPRESSION)

LVALUE -> ident LVALUELOOP

LVALUELOOP -> [EXPRESSION] LVALUELOOP
| ϵ

VARDECL' -> ϵ
| VARDECLLOOP

ATRIBSTAT' -> EXPRESSION
| ALLOCEXPRESSION

IFSTAT' -> else STATEMENT

| ϵ

STATELIST' -> STATELIST
| ϵ

ALLOCEXPRESSION' -> int EXPRESSIONLOOP
| float EXPRESSIONLOOP

| string EXPRESSIONLOOP
EXPRESSIONLOOP' -> EXPRESSIONLOOP
| ϵ

EXPRESSION' -> ϵ
| < NUMEXPRESSION
| > NUMEXPRESSION
| <= NUMEXPRESSION
| >= NUMEXPRESSION
| == NUMEXPRESSION
| != NUMEXPRESSION

4. **Faça CCC-2019-2 ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que CCC-2019-2 está em LL(1) (você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em sala de aula).**

   Program -> STATEMENT
   | **ϵ**
   STATEMENT -> VARDECL ;
   STATEMENT -> ATRIBSTAT ;
   STATEMENT -> PRINTSTAT ;
   STATEMENT -> READSTAT ;
   STATEMENT -> RETURNSTAT ;
   STATEMENT -> IFSTAT ;
   STATEMENT -> FORSTAT ;
   STATEMENT -> { STATELIST }
   STATEMENT -> break ;
   STATEMENT -> ;
   VARDECL -> string ident VARDECL'
   VARDECL -> float ident VARDECL'
   VARDECL -> int ident VARDECL'
   VARDECLLOOP -> [int_constant] VARDECLLOOP
   VARDECLLOOP -> **ϵ**
   ATRIBSTAT -> LVALUE = ATRIBSTAT'
   PRINTSTAT -> print EXPRESSION
   READSTAT -> read LVALUE
   RETURNSTAT -> return
   IFSTAT -> if (EXPRESSION) STATEMENT IFSTAT'
   FORSTAT -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT
   STATELIST -> string ident VARDECL' ; STATELIST'
   STATELIST -> float ident VARDECL' ; STATELIST'
   STATELIST -> int ident VARDECL' ; STATELIST'
   STATELIST -> LVALUE = ATRIBSTAT' ; STATELIST'

STATELIST -> print EXPRESSION ; STATELIST'
STATELIST -> read LVALUE ; STATELIST'
STATELIST -> return ; STATELIST'
STATELIST -> if (EXPRESSION) STATEMENT IFSTAT' ; STATELIST'
STATELIST -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT ; STATELIST'
STATELIST -> { STATELIST } STATELIST'
STATELIST -> break ; STATELIST'
STATELIST -> ; STATELIST'
ALLOCEXPRESSION -> new ALLOCEXPRESSION'
EXPRESSIONLOOP -> [EXPRESSION] EXPRESSIONLOOP'
EXPRESSION -> NUMEXPRESSION EXPRESSION'
NUMEXPRESSION -> TERM TERMLOOP
TERMLOOP -> + TERM TERMLOOP
TERMLOOP -> - TERM TERMLOOP
TERMLOOP -> **ϵ**
TERM -> UNARYEXPR UNARYEXPRLOOP
UNARYEXPRLOOP -> * UNARYEXPR UNARYEXPRLOOP
UNARYEXPRLOOP -> / UNARYEXPR UNARYEXPRLOOP
UNARYEXPRLOOP -> % UNARYEXPR UNARYEXPRLOOP
UNARYEXPRLOOP -> **ϵ**
UNARYEXPR -> + FACTOR
UNARYEXPR -> - FACTOR
UNARYEXPR -> FACTOR
FACTOR -> int_constant
FACTOR -> float_constant
FACTOR -> string_constant
FACTOR -> null
FACTOR ->LVALUE
FACTOR -> (EXPRESSION)
LVALUE -> ident LVALUELOOP
LVALUELOOP -> [EXPRESSION] LVALUELOOP
LVALUELOOP -> **ϵ**
VARDECL' -> [int_constant] VARDECLLOOP
VARDECL' -> **ϵ**
ATRIBSTAT' -> + FACTOR UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> - FACTOR UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> int_constant UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> float_constant UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> string_constant UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> null UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> ident LVALUELOOP UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> (EXPRESSION) UNARYEXPRLOOP TERMLOOP EXPRESSION'
ATRIBSTAT' -> new ALLOCEXPRESSION'
IFSTAT' -> else STATEMENT
IFSTAT' -> **ϵ**
STATELIST' -> string ident VARDECL' ; STATELIST'
STATELIST' -> float ident VARDECL' ; STATELIST'
STATELIST' -> int ident VARDECL' ; STATELIST'
STATELIST' -> ident LVALUELOOP = ATRIBSTAT' ; STATELIST'
STATELIST' -> print EXPRESSION ; STATELIST'
STATELIST' -> read LVALUE ; STATELIST'
STATELIST' -> return ; STATELIST'
STATELIST' -> if (EXPRESSION) STATEMENT IFSTAT' ; STATELIST'
STATELIST' -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT ; STATELIST'
STATELIST' -> { STATELIST } STATELIST'

STATELIST' -> break ; STATELIST'
STATELIST' -> ; STATELIST'
STATELIST' -> **ϵ**
ALLOCEXPRESSION' -> int EXPRESSIONLOOP
ALLOCEXPRESSION' -> float EXPRESSIONLOOP
ALLOCEXPRESSION' -> string EXPRESSIONLOOP
EXPRESSIONLOOP' -> [EXPRESSION] EXPRESSIONLOOP'
EXPRESSIONLOOP' -> **ϵ**
EXPRESSION' -> **ϵ**
EXPRESSION' -> < NUMEXPRESSION
EXPRESSION' -> > NUMEXPRESSION
EXPRESSION' -> <= NUMEXPRESSION
EXPRESSION' -> >= NUMEXPRESSION
EXPRESSION' -> == NUMEXPRESSION
EXPRESSION' -> != NUMEXPRESSION

Tabela de reconhecimento Sintático está localizado na pasta anexos do Zip com o nome ‘tabeladereconhecimentosintático.png’ aplicar bastante zoom na imagem, pois a tabela é muito grande.

5. **Se não usou ferramenta, uma descrição da implementação do analisador sintático (Usou uma tabela de reconhecimento sintático para gramáticas em LL(1)? Se não, então o que foi usado?):**

A implementação do analisador léxico para a gramática CCC-2019-2, foi feita na linguagem Java,sem o uso de ferramentas que constrõem um analisador léxico.

As principais classes da implementação:

1.Tool- nessa classe foi implementada a tabela tabela de reconhecimento sintático para a nossa gramática em uma matriz de array.
O método ‘String analise_sintatica(ArrayList<Token> tokens) ‘ se baseia em uma estrutura de dados pilha,que dado uma iteração de leitura da entrada executa a ação prevista na tabela de reconhecimento sintático,e consome a entrada quando necessário.Ao final das iterações se a entrada e pilha forem vazias significa que não foi encontrado nenhum erro sintático.Quando dado uma entrada não existe ação prevista na tabela ocorre um erro sintático que é informado.