

INE5426 - Construção de Compiladores Analisador Léxico e Analisador Sintático

Florianópolis, 08/10/2019

João Pedro Santana (15204129)

Stefano Bergamini Poletto (16100745)

João Vitor Cardoso (16200647)

Gustavo Ferreira Guimarães (11200638)

1- Analisador Léxico:

1.1 Identificação dos tokens

O processo de converter uma sequência de caracteres em uma sequência de tokens é chamado de “Tokenização”. Este processo é a primeira etapa da criação do “FrontEnd” de um compilador (análise do código fonte para criar uma representação interna do programa).

Os tokens identificados foram:

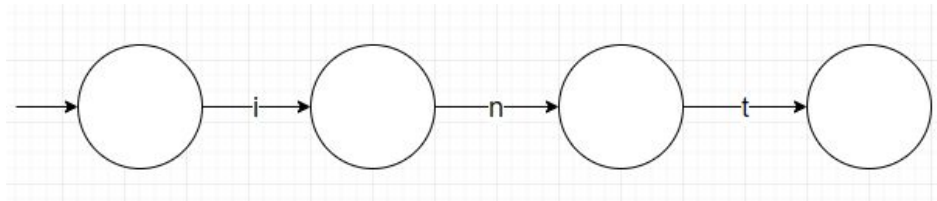
```
INT: 'int';
STRING: 'string';
FLOAT: 'float';
BREAK: 'break';
ENDLINE : ' ';
EQUAL: '=';
PRINT: 'print';
CHAVEA: '{';
CHAVEF: '}';
COLCHA: '[';
COLCHF: ']';
READ: 'read';
RETURN: 'return';
PARENTEA: '(';
PARENTEF: ')';
IF: 'if';
ELSE: 'else';
FOR: 'for';
NEW: 'new';
COMPARADORES: '<' | '>' | '<=' | '>=' | '==' | '!=';
MAISOUMENOS: '+' | '-';
MDP: '*' | '/' | '%';
NULL: 'null';
IDENT: ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
INT_CONSTANT: '0' | ('1'..'9')('0'..'9')*;
FLOAT_CONSTANT: ( '0' | ('1'..'9')('0'..'9')* )'.'('0'..'9')+;
STRING_CONSTANT: ""('a'..'z'|'A'..'Z'|'_'|'0'..'9'| CESPECIAL)*"";
WS : [ \r\t\n ]+ -> skip;
```

1.2 Construção do Diagrama de Transição

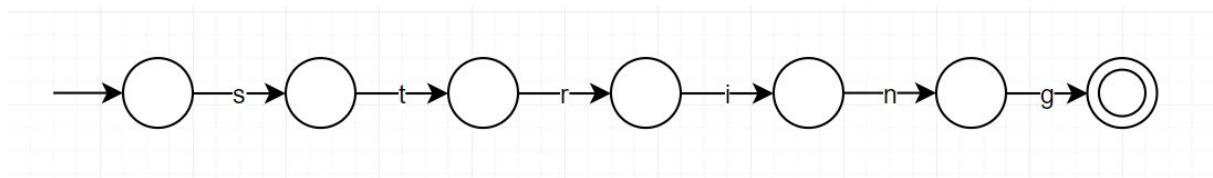
Os diagramas de transição são um tipo especial de fluxogramas usados para análise de linguagens. Nesses diagramas os círculos representam os estados e as setas representam a condição necessária para uma mudança de estado ocorrer.

Estes são os diagramas de transição para os tokens encontrados:

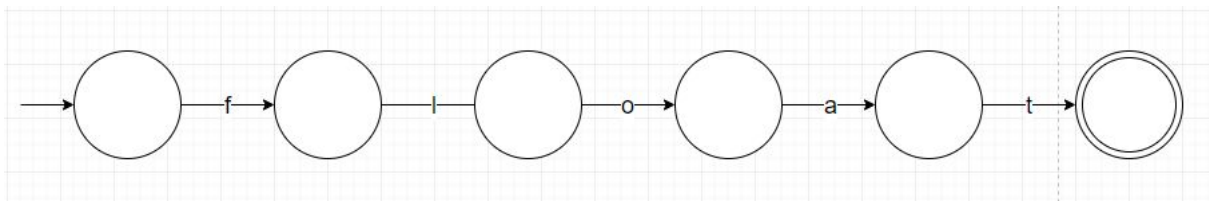
INT:



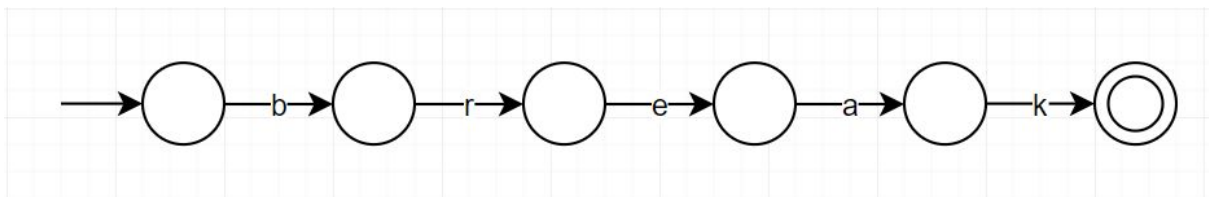
STRING:



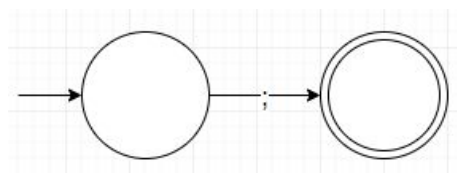
FLOAT:



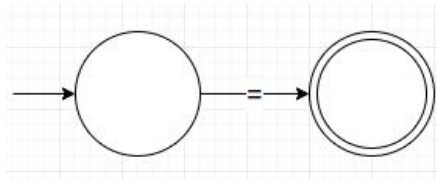
BREAK:



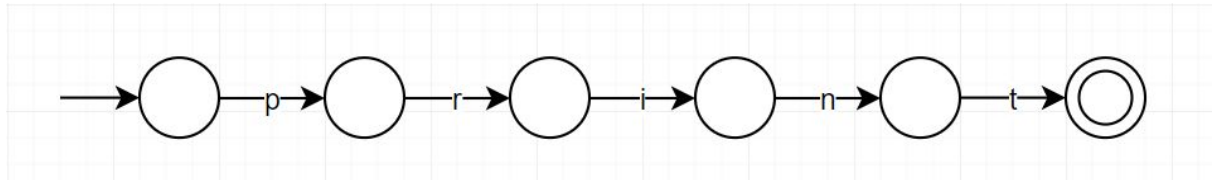
ENDLINE:



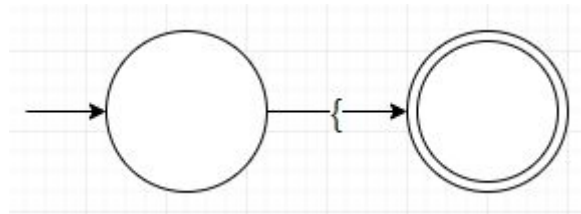
EQUAL:



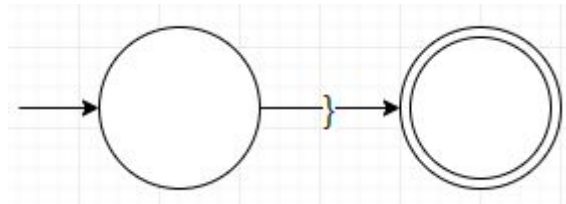
PRINT:



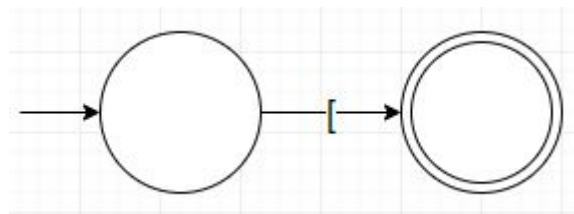
CHAVEA:



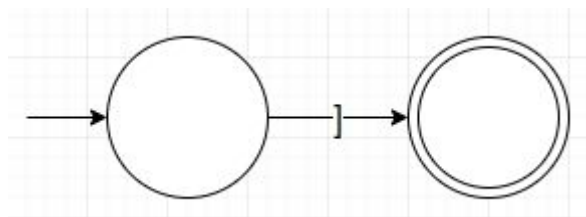
CHAVEF:



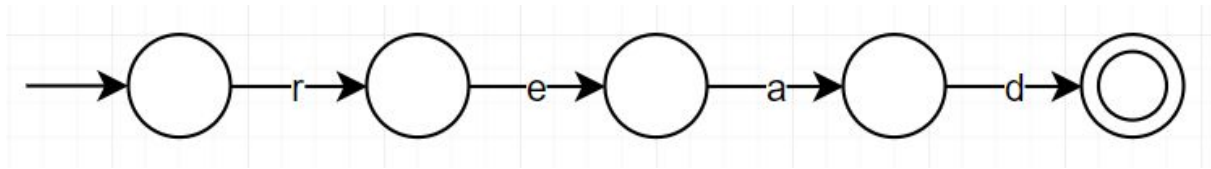
COLCHA:



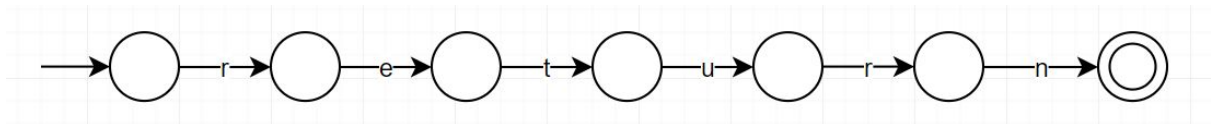
COLCHF:



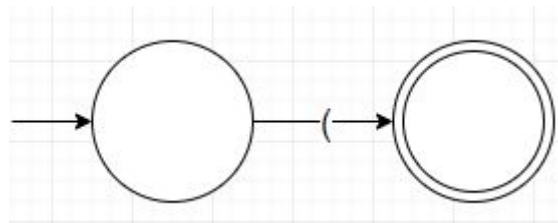
READ:



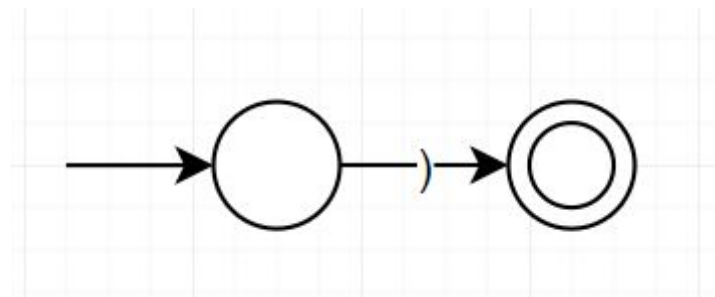
RETURN:



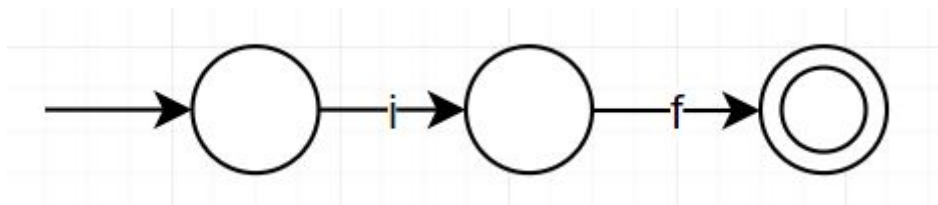
PARENTEA:



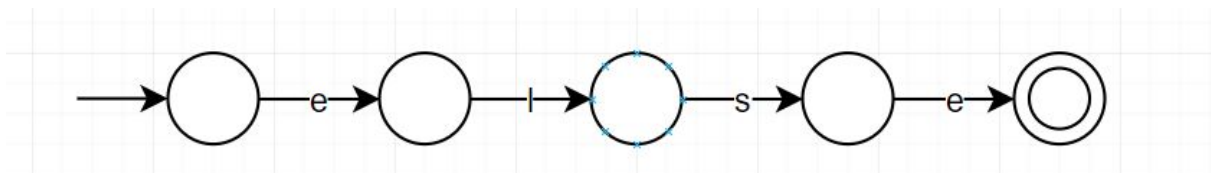
PARENTEF:



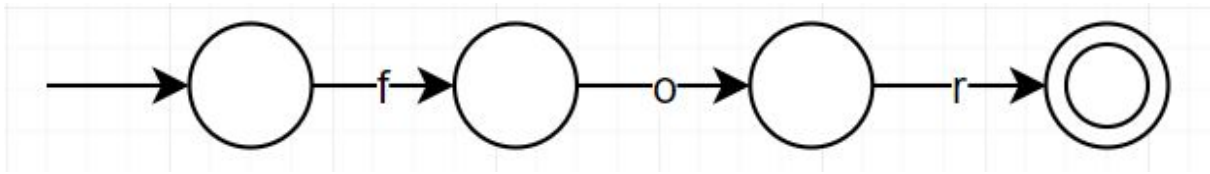
IF:



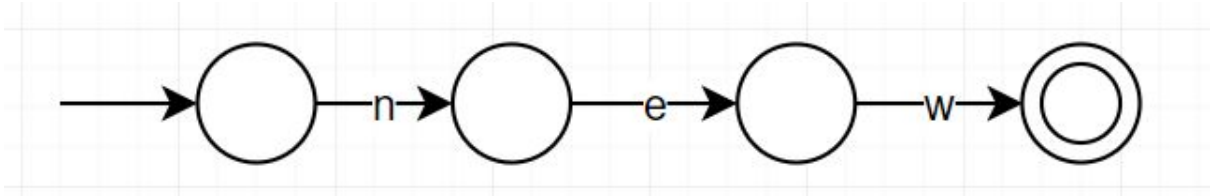
ELSE:



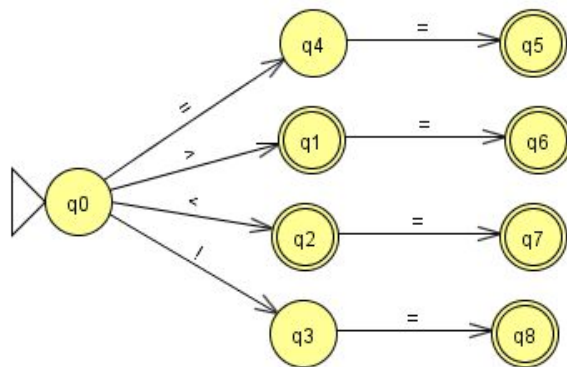
FOR:



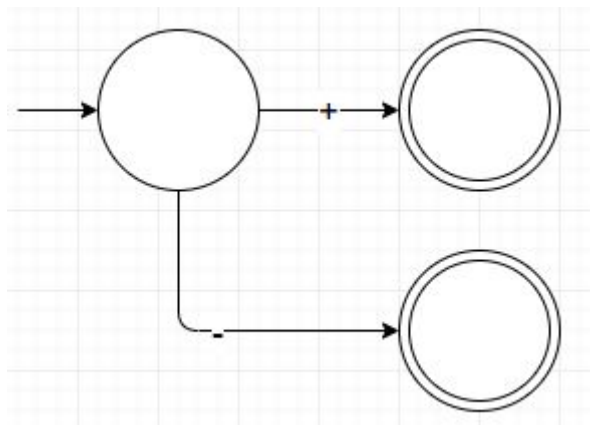
NEW:



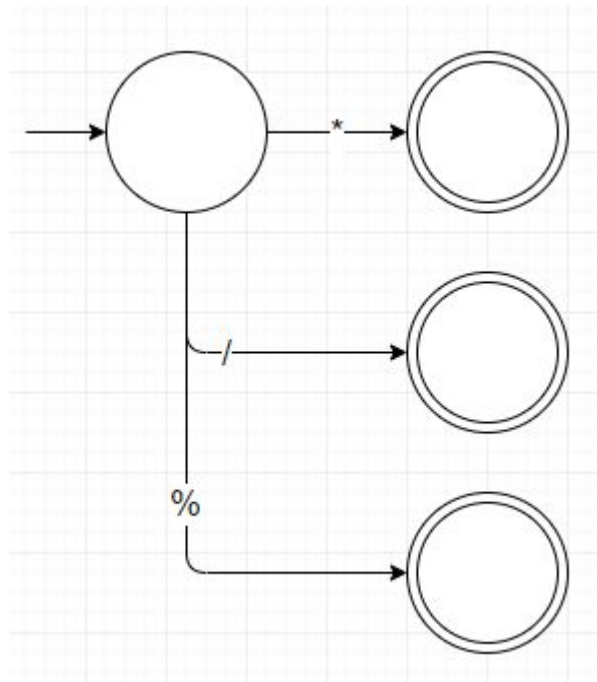
COMPARADORES:



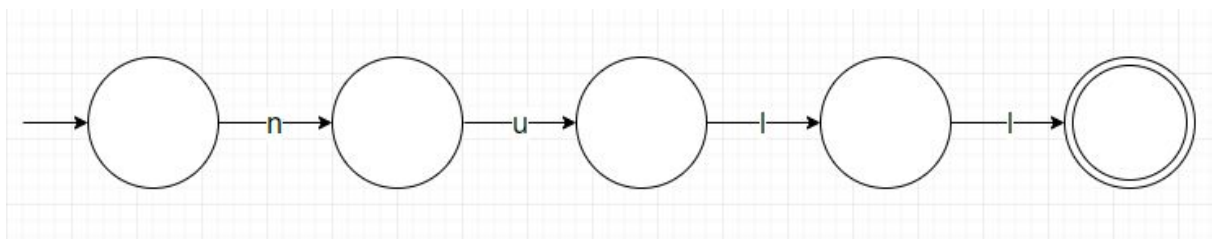
MAISOU MENOS:



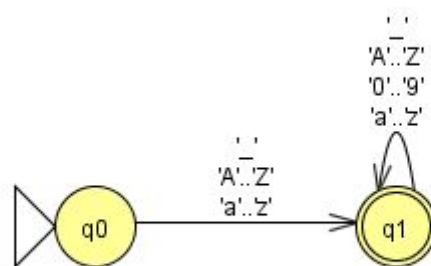
MDP:



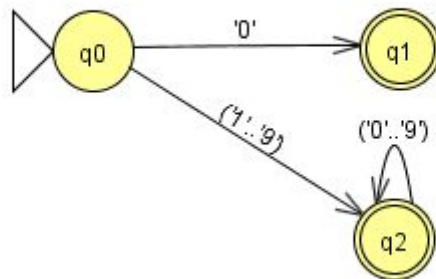
NULL:



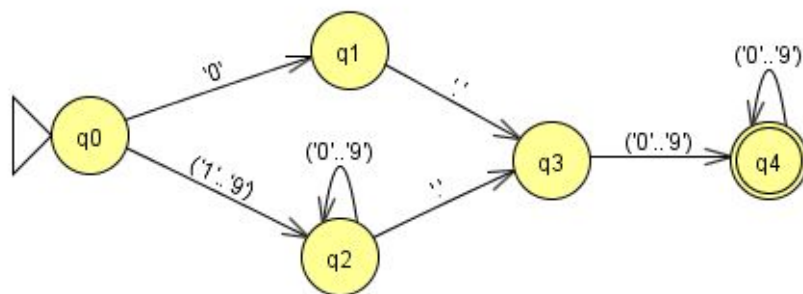
IDENT:



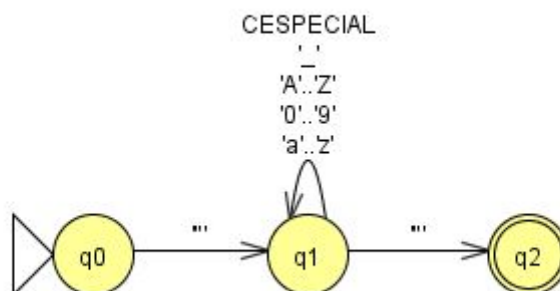
INT_CONSTANT:



FLOAT_CONSTANT:

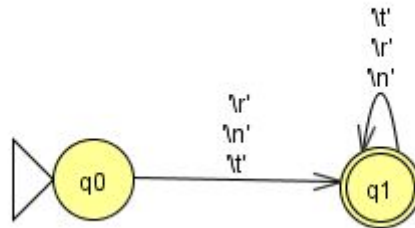


STRING_CONSTANT:



(obs: CESPECIAL caracteres especiais)

WS:



1.3 Ferramenta Antlr - Entradas e saídas para o analisador léxico

A ferramenta Antlr foi utilizada para realização desta etapa do trabalho. Para utilizar esta ferramenta foi necessário a construção um arquivo em formato “.g4” a partir da gramática CC-2019-2 dada pelo professor da disciplina.

Esse arquivo segue o seguinte padrão:

- Terminais representados com todas as letras maiúsculas;
- Não-Terminais representados com todas as letras minúsculas
- Produções representadas da forma -> nomeProdução: produção1 | produção2 | ... | produçãoX | ;

```

program: statement
      |
      ;

statement: vardecl ENDLINE
          | atribstat ENDLINE
          | printstat ENDLINE
          | readstat ENDLINE
          | returnstat ENDLINE
          | ifstat
          | forstat
          | CHAVEA statelist CHAVEF
          | BREAK ENDLINE
          | ENDLINE
          ;

vardecl: a IDENT b;

a:  INT
   | FLOAT
   | STRING
   ;
b:  COLCHA INT_CONSTANT COLCHF
   | COLCHA INT_CONSTANT COLCHF b
   ;

```

(Screenshot de um trecho de uma gramática no formato “.g4”)

Com a gramática CCC-2019-2 reformulada para o formato “.g4”, pudemos executar o Antlr em cima desse arquivo para gerarmos documentos de extensão “.py” e documentos próprios do Antlr.

Comando para gerar os arquivos:

```
antlr4 -Dlanguage=Python3 gramaticaAntlr.g4
```

obs: executar este comando dentro do diretório gramatics/gramaticas/

Esse comando gera arquivos com os tokens da gramática (nomes literais, nomes simbólicos e valores), nomes das regras e 3 programas em python para a análise léxica e sintática de uma código fonte.

Para realizarmos a interpretação de uma gramática, deve-se executar o arquivo “AL.py” enviando como argumento da chamada desse programa em Python o código fonte que se deseja ser interpretado. A saída é apresentada diretamente no console informando uma lista de tokens.

Exemplo de comando para execução do programa “AL.py”:

python3 AL.py ./codigos/codigo_1.ccc

obs: comando executado dentro da pasta gramatics

Caso tenha Sucesso em tudo

```
python3 ./gramatics/AL.py ./gramatics/codigos/codigo_1.ccc
```

Index	Token	Line/Column	Type
0	{	(1, 0)	CHAVEA
1	{	(2, 0)	CHAVEA
2	int	(3, 0)	INT
3	a	(3, 4)	IDENT
4	;	(3, 5)	ENDLINE
5	int	(4, 0)	INT
6	b	(4, 4)	IDENT
7	;	(4, 5)	ENDLINE
8	int	(5, 0)	INT
9	c	(5, 4)	IDENT
10	;	(5, 5)	ENDLINE
11	int	(6, 0)	INT
12	d	(6, 4)	IDENT
13	;	(6, 5)	ENDLINE
14	int	(7, 0)	INT
15	e	(7, 4)	IDENT
16	;	(7, 5)	ENDLINE
17	int	(8, 0)	INT
18	max	(8, 4)	IDENT
19	;	(8, 7)	ENDLINE

(Resultado da execução do programa “AL.py”)

Caso Tenha um defeito a qual não reconhece um token ele printa uma mensagem de em qual linha e qual caractere não foi reconhecido.

```
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$ make RUN_AL_1
python3 ./gramatics/AL.py ./gramatics/codigos/codigo_1.ccc
line 3:4 token recognition error at: '?'
```

Index	Token	Line/Column	Type
0	{	(1, 0)	CHAVEA
1	{	(2, 0)	CHAVEA
2	int	(3, 0)	INT
3	;	(3, 5)	ENDLINE
4	int	(4, 0)	INT
5	b	(4, 4)	IDENT
6	;	(4, 5)	ENDLINE
7	int	(5, 0)	INT
8	c	(5, 4)	IDENT
9	;	(5, 5)	ENDLINE
10	int	(6, 0)	INT
11	d	(6, 4)	IDENT
12	;	(6, 5)	ENDLINE
13	int	(7, 0)	INT
14	e	(7, 4)	IDENT
15	;	(7, 5)	ENDLINE
16	int	(8, 0)	INT
17	max	(8, 4)	IDENT

2- Analisador Sintático:

2.1 Gramática CC-2019-2 (Forma convencional)

PROGRAM -> STATEMENT | &

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

VARDECL -> A ident B

A -> int | float | string

B -> [int_constant]B | &

ATRIBSTAT -> LVALUE = EXPRESSION | LVALUE = ALLOCEXPRESSION

PRINTSTAT -> print EXPRESSION

READSTAT -> read LVALUE

RETURNSTAT -> return

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

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

STATELIST -> STATEMENT | STATEMENT STATELIST

ALLOCEXPRESSION -> new A C

C -> [EXPRESSION] | [EXPRESSION]C

EXPRESSION -> NUMEXPRESSION | NUMEXPRESSION D NUMEXPRESSION

D -> < | > | <= | >= | == | !=

NUMEXPRESSION -> TERM N

N -> MAISOUMENOS TERM | MAISOUMENOS TERM N | &

MAISOUMENOS -> + | -

TERM -> UNARYEXPR E

E -> F UNARYEXPR | F UNARYEXPR E | &

F -> * | \ | %

UNARYEXPR-> MAISOU MENOS FACTOR | FACTOR

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

LVALUE-> ident G

G-> [EXPRESSION]G | &

2.2 Gramática CC-2019-2 (Recursão à esquerda)

A gramática CC-2019-2 não possui recursão à esquerda, porque nenhum dos símbolos não terminais levam a si mesmos direta ou indiretamente através das produções.

Para chegar a essa conclusão foi sendo checada cada produção da gramática procurando casos diretos e indiretos.

DIRETO: $A \rightarrow Aa$, para 'a' contido em V^*

INDIRETA: $S \rightarrow Aa$ e $A \rightarrow Sb$, para a^b contido em V^*

2.3 Gramática CC-2019-2 (Fatorada à esquerda)

A fatoração à esquerda vem da ideia básica que caso não exista uma decisão clara para duas ou mais produções alternativas se expande para um novo não-terminal, fazendo assim a decisão depois realizando a escolha certa.

Em geral se ocorre um caso como por exemplo:

$A \rightarrow ab1 \mid ab2$

Nesse caso se começa a derivar o "a" e não se sabe se vai ser realizado o ab1 ou o ab2, por isso se expande A para uma nova produção resultando em:

$A \rightarrow aA'$

$A' \rightarrow b1 \mid b2$

Com isso de base foi realizada a fatoração à esquerda da gramática.

A gramática CC-2019-2 não está fatorada a esquerda, a sua forma fatorada a esquerda ficará assim:

PROGRAM -> STATEMENT|&

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

VARDECL-> A ident B

A -> int | float | string

B -> [int_constant]B | &

ATRIBSTAT-> LVALUE = **H**

H-> EXPRESSION | ALLOCEXPRESSION

PRINTSTAT-> print EXPRESSION

READSTAT-> read LVALUE

RETURNSTAT-> return

IFSTAT-> if(EXPRESSION) STATEMENT **Z**

Z-> else STATEMENT | &

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

STATELIST-> STATEMENT **J**

J-> STATELIST | &

ALLOCEXPRESSION-> new A C

C-> [EXPRESSION]**K**

K-> C | &

EXPRESSION-> NUMEXPRESSION **L**

D-> < | > | <= | >= | == | !=

L-> D NUMEXPRESSION | &

NUMEXPRESSION-> TERM **N**

N-> MAISOUMENOS TERM M | &

MAISOUMENOS-> + | -

M-> N | &

TERM-> UNARYEXPR **E**

E-> F UNARYEXPR O | &

F-> * | \ | %

O-> E | &

UNARYEXPR-> MAISOUMENOS FACTOR | FACTOR

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

LVALUE-> ident **G**

G-> [EXPRESSION] G | &

(Obs: Todas as mudanças estão em **negrito**)

2.4 Gramática CC-2019-2 (Transformar em LL(1))

Uma gramática é dita LL(1), sempre que $A \rightarrow a \mid b$ forem duas produções distintas de G e seguem as seguintes regras:

1. “a” e “b” não derivam, ao mesmo tempo, cadeias começando pelo mesmo terminal “a”, qualquer que seja “a” (Gramática fatorada);
2. No máximo um dos dois, “a” ou “b”, derive a cadeia vazia;
3. Se $b \rightarrow * \&$, então “a” não deriva qualquer cadeia começando por um terminal em FOLLOW(A).

Com isso em mente se fez o cálculo dos FIRSTS e FOLLOWS.

Primeiro calculamos os FIRSTS.

first(PROGRAM) = {&, {, break, ;, int, float, string, print, return, for, ident, if, read}

first(STATEMENT) = { {, break, ;, int, float, string, print, return, for, ident, if, read}

first(A) = {int, float, string}

first(B) = {[, &}

first(PRINTSTAT) = {print}

first(READSTAT) = {read}

first(RETURNSTAT) = {return}

first(IFSTAT) = {if}

first(Z) = {else, &}

first(FORSTAT) = {for}

first(J) = {&, {, break, ;, int, float, string, print, return, for, ident, if, read}

first(ALLOEXPRESSION) = {new}

first(C) = {[}

first(K) = {&, [}

first(D) = {<, >, <=, >=, ==, !=}

first(L) = {&, <, >, <=, >=, ==, !=}

first(N) = {&, +, -}

first(MAISOU MENOS) = {+, -}

first(M) = {&, +, -}

first(E) = {&, *, \, %}

first(F) = {*, \, %}

first(O) = {&, *, \, %}

first(FACTOR) = {int_constant, float_constant, string_constant, null, lvalue, {}

first(LVALUE) = {ident}

```

first(G) = {[, &}
first(VARDECL) = {int, float, string}
first(ATRIBSTAT) = {ident}
first(UNARYEXPR) = {+, -, int_constant, float_constant, string_constant, null,
lvalue, {}
first(TERM) = {+, -, int_constant, float_constant, string_constant, null, lvalue,
{}
first(NUMEXPRESSION) = {+, -, int_constant, float_constant, string_constant,
null, lvalue, {}
first(STATELIST) = {{, break, :, int, float, string, print, return, for, ident, if, read}
first(EXPRESSION) = {+, -, int_constant, float_constant, string_constant, null,
lvalue, {}
first(H) = {+, -, int_constant, float_constant, string_constant, null, lvalue, (,
new}

```

Realizado o cálculo dos FIRSTS foi feito os FOLLOWS:

```

follow(PROGRAM) = {$}
follow(STATEMENT) = { &, {, break, :, int, float, string, print, return, for, ident,
if, read, else, $ }
follow(VARDECL) = {;}
follow(A) = {[, ident}
follow(B) = {;}
follow(ATRIBSTAT) = {;, )}
follow(H) = {;, )}
follow(PRINTSTAT) = {;}
follow(READSTAT) = {;}
follow(RETURNSTAT) = {;}
follow(IFSTAT) = {&, {, break, :, int, float, string, print, return, for, ident, if,
read, else, $}
follow(Z) = {&, {, break, :, int, float, string, print, return, for, ident, if, read, else,
$}
follow(FORSTAT) = {&, {, break, :, int, float, string, print, return, for, ident, if,
read, else, $}
follow(STATELIST) = {"}"
follow(J) = {"}"
follow(ALLOCEXPRESSION) = {;, )}
follow(C) = {;, )}
follow(K) = {;, )}
follow(EXPRESSION) = {[, ), ;}
follow(D) = {+, -, int_constant, float_constant, string_constant, null, lvalue, {}
follow(L) = {[, ), ;}

```


$\text{follow}(\text{NUMEXPRESSION}) = \{ \&, <, >, <=, >=, ==, !=, ;,],) \}$
 $\text{follow}(\text{N}) = \{ \&, <, >, <=, >=, ==, !=, ;,],) \}$
 $\text{follow}(\text{MAISOUMENOS}) = \{ \text{int_constant}, \text{float_constant}, \text{string_constant}, \text{null}, \text{lvalue}, (, +, - \}$
 $\text{follow}(\text{M}) = \{ \&, <, >, <=, >=, ==, !=, ;,],) \}$
 $\text{follow}(\text{TERM}) = \{ \&, +, - \}$
 $\text{follow}(\text{E}) = \{ \&, +, - \}$
 $\text{follow}(\text{F}) = \{ +, -, \text{int_constant}, \text{float_constant}, \text{string_constant}, \text{null}, \text{lvalue}, (\}$
 $\text{follow}(\text{O}) = \{ \&, +, - \}$
 $\text{follow}(\text{UNARYEXPR}) = \{ \&, *, \backslash, \% \}$
 $\text{follow}(\text{FACTOR}) = \{ \&, *, \backslash, \% \}$
 $\text{follow}(\text{LVALUE}) = \{ =, ; \}$
 $\text{follow}(\text{G}) = \{ =, ; \}$

Com o término desses passos se utilizou deles e foi sendo checado produção por produção para alcançar o resultados desejado, sendo ele a gramática CCC-2019-2 em LL(1).

$\text{PROGRAM} \rightarrow \text{STATEMENT} \mid \&$
 $\text{STATEMENT} \rightarrow \text{VARDECL} \mid \text{ATRIBSTAT} \mid \text{PRINTSTAT} \mid \text{READSTAT} \mid \text{RETURNSTAT} \mid \text{IFSTAT} \mid \text{FORSTAT} \mid \{ \text{STATELIST} \} \mid \text{break} \mid ;$

$\text{VARDECL} \rightarrow \text{A ident B}$
 $\text{A} \rightarrow \text{int} \mid \text{float} \mid \text{string}$
 $\text{B} \rightarrow [\text{int_constant}] \text{B} \mid \&$

$\text{ATRIBSTAT} \rightarrow \text{LVALUE} = \text{H}$
 $\text{H} \rightarrow \text{EXPRESSION} \mid \text{ALLOCEXPRESSION}$

$\text{PRINTSTAT} \rightarrow \text{print EXPRESSION}$
 $\text{READSTAT} \rightarrow \text{read LVALUE}$
 $\text{RETURNSTAT} \rightarrow \text{return}$

$\text{IFSTAT} \rightarrow \text{if}(\text{EXPRESSION}) \text{STATEMENT Z}$
 $\text{Z} \rightarrow \text{else STATEMENT} \mid \&$

$\text{FORSTAT} \rightarrow \text{for}(\text{ATRIBSTAT}; \text{NUMEXPRESSION}; \text{ATRIBSTAT}) \text{STATEMENT}$

$\text{STATELIST} \rightarrow \text{STATEMENT J}$
 $\text{J} \rightarrow \text{STATELIST} \mid \&$

ALLOCEXPRESSION-> new A C
C-> [EXPRESSION]K
K-> C | &

EXPRESSION-> NUMEXPRESSION L
D-> < | > | <= | >= | == | !=
L-> D NUMEXPRESSION | &

NUMEXPRESSION-> TERM N

N-> MAISOU MENOS TERM M | &
MAISOU MENOS-> + | -
M-> N | &

TERM-> UNARYEXPR E
E-> F UNARYEXPR O | &
F-> * | \ | %
O-> E | &

UNARYEXPR-> MAISOU MENOS FACTOR | FACTOR

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

LVALUE-> ident G
G-> [EXPRESSION] G | &

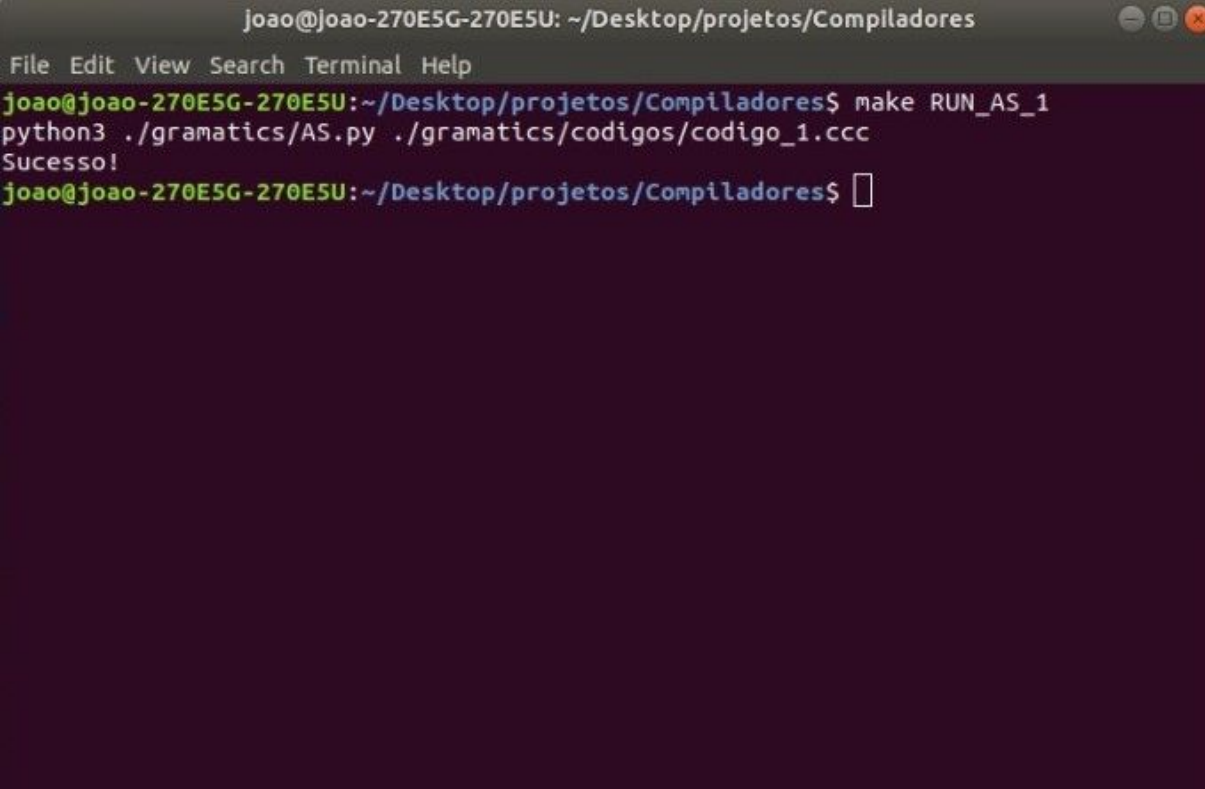
De acordo com as regras citadas anteriormente, a gramática CC-2019-2
fatorada já é uma gramática LL(1).

2.6 Ferramenta Antlr - Entradas e saídas para o analisador sintático

Para o analisador sintático se segue a mesma lógica utilizada para o analisador léxico, Para realizarmos a análise sintática de uma gramática, deve-se executar o arquivo "AS.py" enviando como argumento da chamada desse programa

em Python o código fonte que se deseja ser analisado. A saída é apresentada diretamente no console informando uma mensagem de sucesso ou de erro, como mostra as imagens a seguir.

Em Caso de Sucesso somente uma Mensagem informando isso

A screenshot of a Linux terminal window. The title bar reads 'joao@joao-270E5G-270E5U: ~/Desktop/projetos/Compiladores'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores\$'. The user enters the command 'make RUN_AS_1 python3 ./gramatics/AS.py ./gramatics/codigos/codigo_1.ccc'. The output is 'Sucesso!'. The prompt returns to 'joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores\$' with a cursor.

```
joao@joao-270E5G-270E5U: ~/Desktop/projetos/Compiladores
File Edit View Search Terminal Help
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$ make RUN_AS_1
python3 ./gramatics/AS.py ./gramatics/codigos/codigo_1.ccc
Sucesso!
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$
```

Em caso de Insucesso:

```
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$ make RUN_AS_1
python3 ./gramatics/AS.py ./gramatics/codigos/codigo_1.ccc
Sucesso!
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$ make RUN_AS_1
python3 ./gramatics/AS.py ./gramatics/codigos/codigo_1.ccc
Erro Sintatico
line 10:3
Regra: statelist
Symbolo(Token): ; (ENDLINE)

Forma sentencial:
(program (statement { (statelist { int a ; int b ; int c ; int d ; int e ; int m
ax ; a = 1 ; b = ; c = 3 ; d = 4 ; e = 5 ; max = 100 ; int i ; for ( i = 0 ; i <
max ; i = i + 1 ) { a = a + 1 ; b = b + 1 ; c = c + 1 ; d = d + 1 ; e = e + 1 ;
) })))
joao@joao-270E5G-270E5U:~/Desktop/projetos/Compiladores$
```