Relatório de Projeto

Silvano Junior (12011BCC042)
Thais Damasceno (11721BCC007)
Vitor Yuji (11921BCC021)

08.09.2024

Bacharelado Ciência da Computação - Construção de Compiladores

Etapa 1 - Projeto da Linguagem

Especificação da linguagem

– Definição da gramática livre de contexto (GLC) com as estruturas da linguagem especificada

- Estrutura principal;
- Componentes Básicos;
- Comandos e Estruturas de Controle;
- Condições e Expressões;
- Identificadores e Comentários.

A gramática livre do contexto G é definida como:

G = (V, T, P, S)

Onde:
V (Variáveis/Não-terminais)
T (Terminais)
P (Regras de Produção)
S (Símbolo inicial)

V =
Programa, Bloco, Declaracoes, DeclaracaoVar, Tipo, ListaIds, Comandos, Comando, Atribuicao, ComandoSelecao, ComandoElse, ComandoRepeticao, Condicao, OperadorRelacional, Expressao, Termo, Fator

T =
main, begin, end, int, char, float, if, then, else, while, do, repeat, until, :=, :, ;, ,, ==, !=, <, >, <=, >=, +, -, *, /, **, (,), ID, INT_CONST, FLOAT_CONST, CHAR_CONST, {Comentario}

P = {

Programa → main ID Bloco

Bloco → begin Declaracoes Comandos end

Declaracoes → DeclaracaoVar Declaracoes | ε

DeclaracaoVar → Tipo : ListaIds ;

Tipo → int | char | float

ListaIds → tipo -> ID , ListaIds | tipo -> ID

Comandos → Comando Comandos | ε

Comando → Atribuicao | ComandoSelecao | ComandoRepeticao | Bloco | {Comentario}

Atribuicao → ID := Expressao ;

ComandoSelecao → if (Condicao) then Comando ComandoElse

ComandoElse → else Comando | ε

ComandoRepeticao → while(Condicao) do Comando | repeat Comando until (Condicao) ;

Condicao → Expressao OperadorRelacional Expressao

OperadorRelacional → == | != | < | > | <= | >=

Expressao → Expressao + Termo | Expressao - Termo | Termo

Termo → Termo * Fator | Termo / Fator | Termo ** Fator | Fator

Fator → (Expressao) | ID | INT_CONST | FLOAT_CONST | CHAR_CONST

}

S → Programa

– Identificação dos tokens usados na gramática

- **Apresentar uma tabela com o nome e o tipo de atributo que será retornado (quando aplicável) para cada token**

LEXEMA	NOME DO TOKEN	valor do atributo
main	main	-
begin	begin	-
end	end	-
int	tipo	INT
char	tipo	CHAR
float	tipo	FLOAT
if	if	-
then	then	-
else	else	-
while	while	-
do	do	-
repeat	repeat	-
until	until	-
:	pont	COLON
;	pont	SEMICOLON

,	pont	COMMA
:=	pont	ASSIGN
()	pont	PAREN
->	associacao	ARR

==	relop	EQ
!=	relop	NEQ
<	relop	LT
>	relop	GT
<=	relop	LE
>=	relop	GE
+	ariop	ADD
-	ariop	SUB
*	ariop	MUL
/	ariop	DIV
**	ariop	EXP
Qualquer id	lista_ids	Posição na tabela de símbolos
qualquer número	INT_CONST	Posição na tabela de símbolos
qualquer constante float	FLOAT_CONST	Posição na tabela de símbolos
Qualquer constante char	CHAR_CONST	Posição na tabela de símbolos
Qualquer ws	-	-
Qualquer comentário	-	-

– Definição dos padrões (expressões regulares) de cada token (inclusive os tokens especiais)

ID → [A-Za-z_][A-Za-z0-9_]*

relop → **==** | **!=** | **<** | **>** | **<=** | **>=**

ariop → + | - | * | / | **

pont → := | : | ; | , | (|)

tipo → INT | CHAR | FLOAT

INT_CONST → -?([0-9]{1,5})

FLOAT_CONST → [0-9]+(.[0-9]+)?([Ee][-]?[0-9]+)?

CHAR_CONST →'[ID]'

associacao → ->

lista_ids → tipo->ID (,ID)*

MAIN → main

BEGIN → begin

END → end

IF → if

THEN → then

ELSE → else

WHILE → while

DO → do

REPEAT → repeat

UNTIL → until

COMMENT → \{[^}]*\}

ws → (‘ ‘ | \t | \n)*

Etapa 2 - Análise Léxica

- **Gerar um diagrama de transição para cada token**

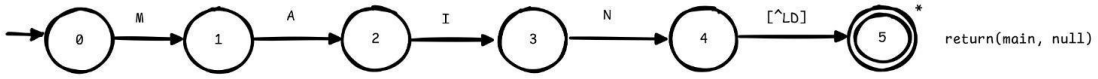
Para facilitar o entendimento dos diagramas:

letra_ → [A – Z a – z _]

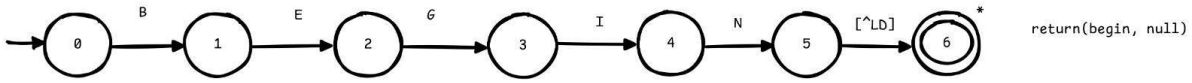
digito → [0 – 9]

[^LD] = [^letra_digito]
[D] = digito

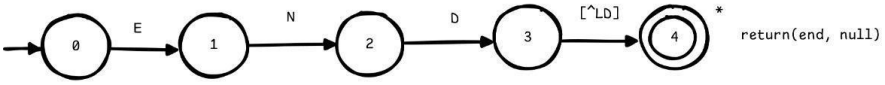
token main



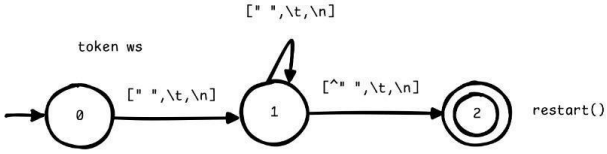
token begin



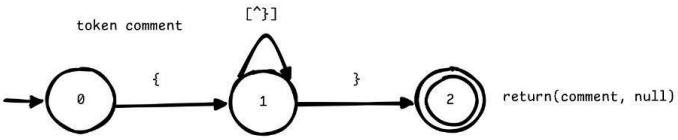
token end



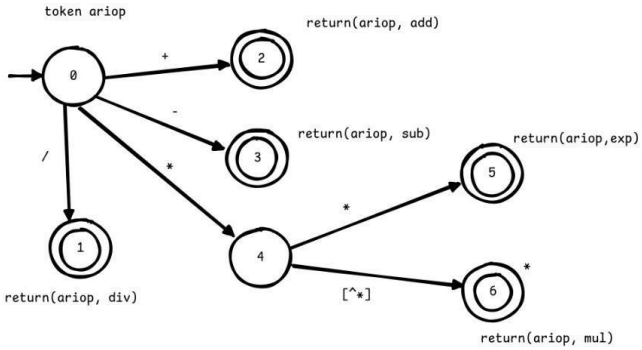
token ws



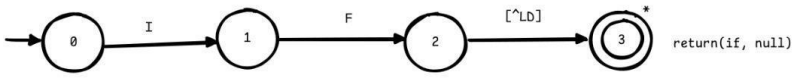
token comment



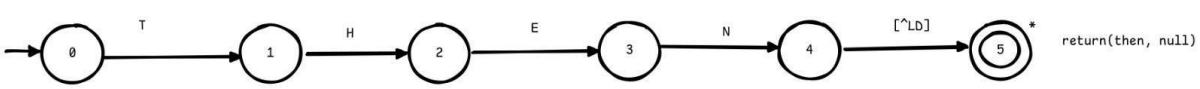
token ariop

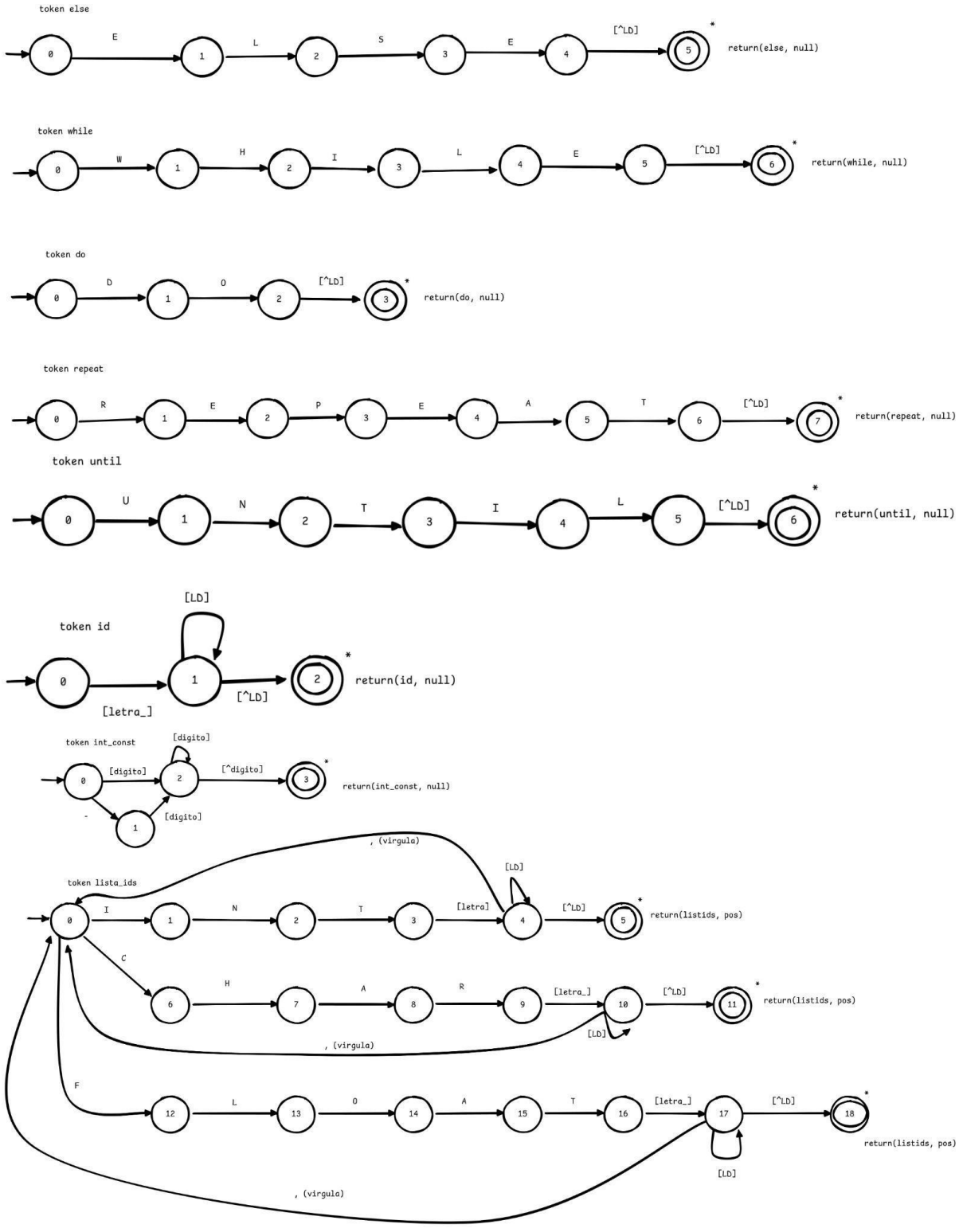


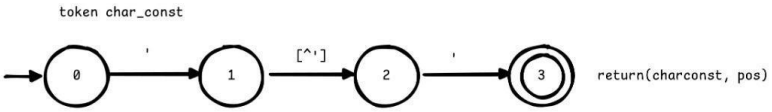
token if

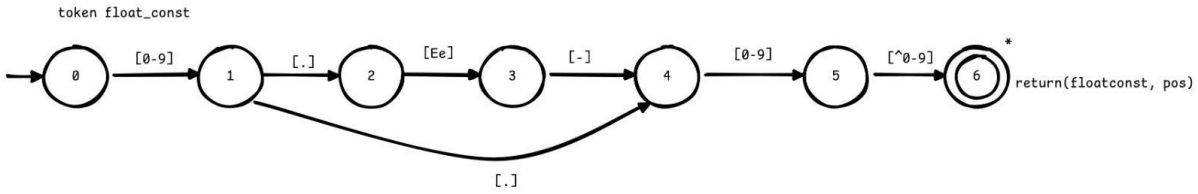


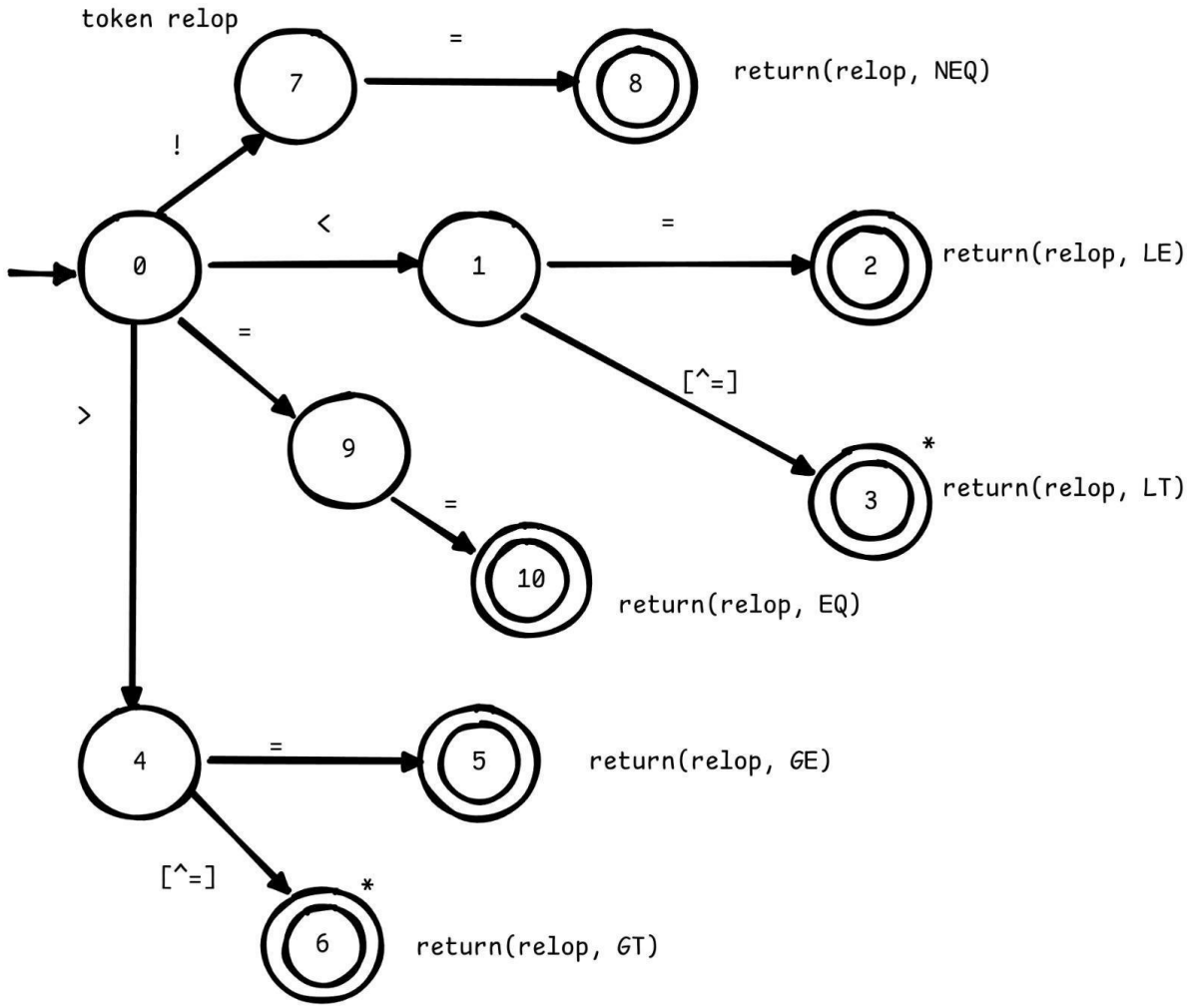
token then



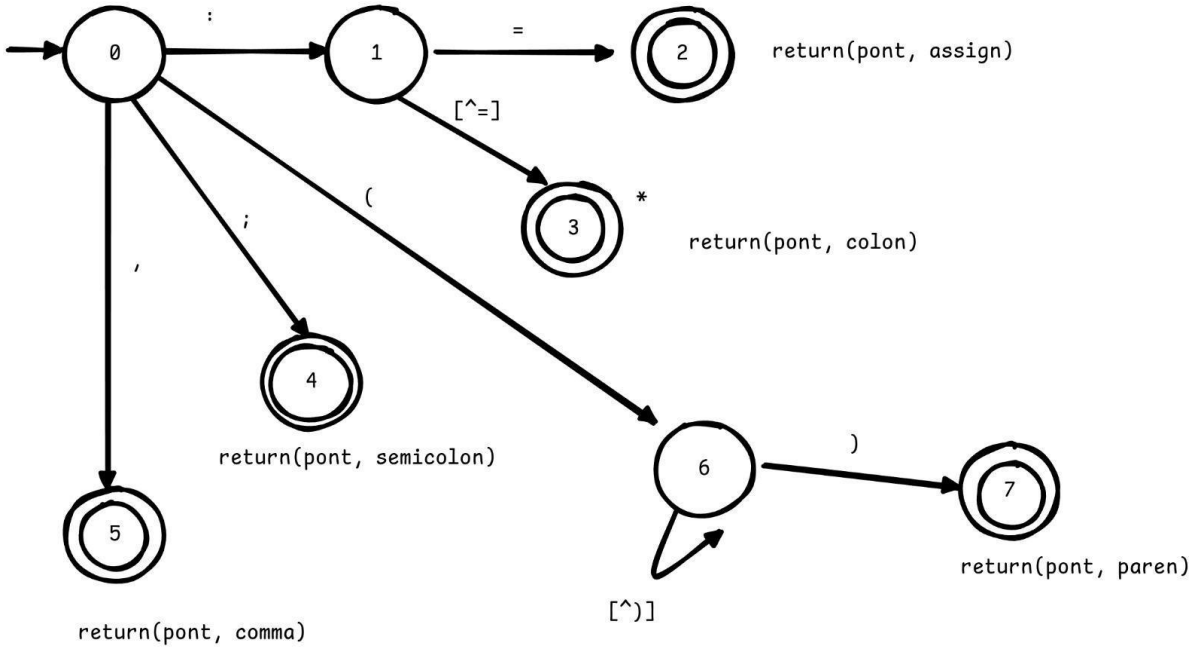

token else
E
L
S
E
[^LD]
return(else, null)
token while
W
H
I
L
E
[^LD]
return(while, null)
token do
D
O
[^LD]
return(do, null)
token repeat
R
E
P
E
A
T
[^LD]
return(repeat, null)
token until
U
N
T
I
L
[^LD]
return(until, null)
token id
[LD]
[letra_]
[^LD]
return(id, null)
token int_const
[digito]
[digito]
[^digito]
return(int_const, null)
-
[digito]
token lista_ids
, (virgula)
[LD]
I
N
T
[letra]
[^LD]
return(listids, pos)
C
H
A
R
[letra_]
[^LD]
return(listids, pos)
[LD]
, (virgula)
F
L
O
A
T
[letra_]
[^LD]
return(listids, pos)
[LD]
, (virgula)


token char_const
0
'
1
[^']
2
'
3
return(charconst, pos)

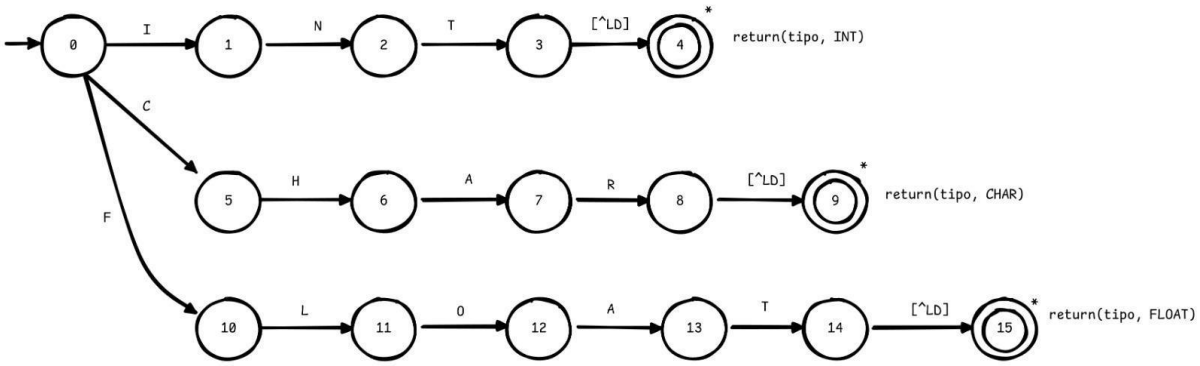

token float_const
0
[0-9]
1
[.]
2
[Ee]
3
[-]
4
[0-9]
5
[^0-9]
6
*
return(floatconst, pos)
[.]


token relop
7
=
8
return(relop, NEQ)
!
0
<
1
=
2
return(relop, LE)
=
[^=]
>
9
3
*
return(relop, LT)
=
10
return(relop, EQ)
4
=
5
return(relop, GE)
[^=]
*
6
return(relop, GT)

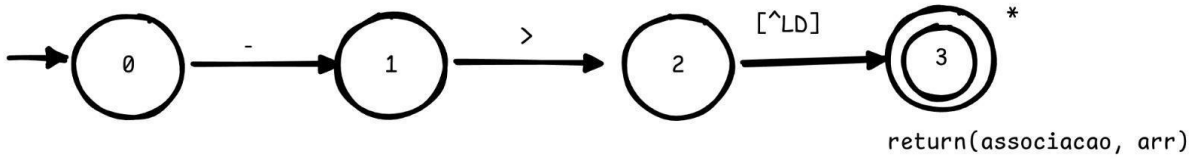
token pont



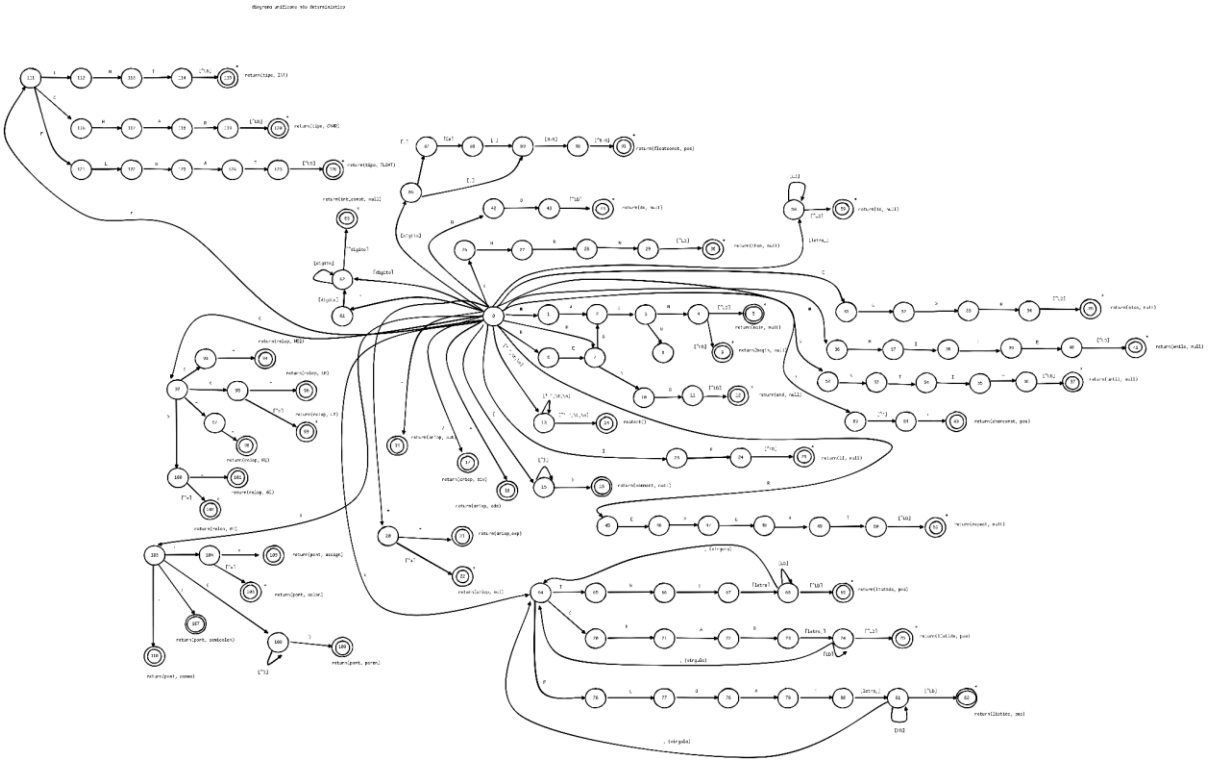
token tipo



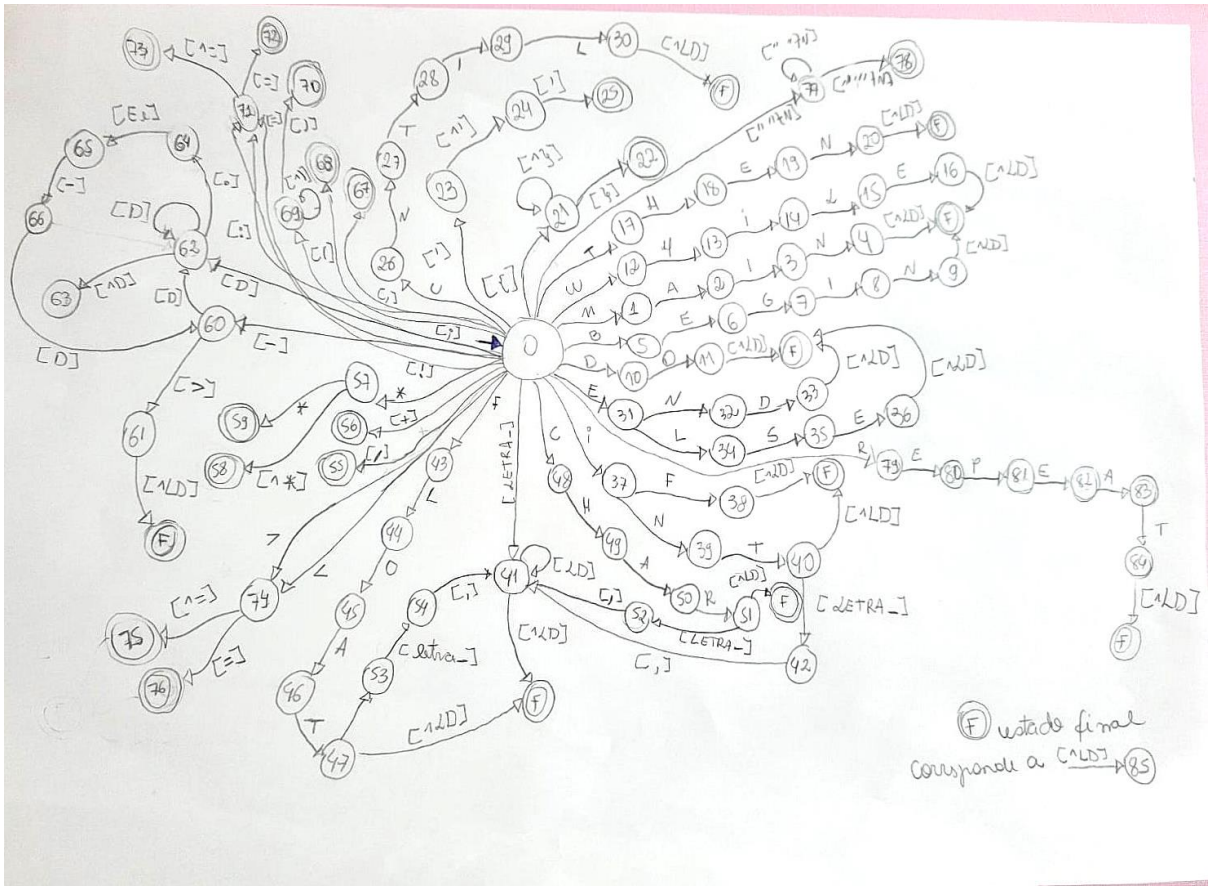
token arr



- **Unificá-los em um diagrama não determinístico**



- **Convertê-lo em um diagrama de transição determinístico**



A partir da entrada [-] é possível ligar os diagramas ariop, arr, id e int_const

A partir dos tipos int, char e float do é possível ligar os diagramas float_const, int_const, id e listaIds.
A partir de ‘e’ end e else
A partir de ‘i’ if e int (de tipo)
A partir da entrada [=] é possível ligar os diagramas relop e pont

Etapa 3 - Análise Sintática

– Conversão da GLC da linguagem para LL(1)

- **Remocação de recursão a esquerda**
- **Tratamento de ambiguidades**

Programa → main ID Bloco

Bloco → begin Declaracoes Comandos end

Declaracoes → Tipo -> ListaIds ;
| ε

Tipo → int
| char
| float

ListaIds → ID ListaIds'

ListaIds' → , ListaIds
| ε

Comandos → Comando Lista_comandos

Lista_comandos → Comando Lista_comandos
| ε

Comando → Atribuicao
| ComandoSelecao
| ComandoRepeticao
| Bloco

Atribuicao → ID := Expressao ;

ComandoSelecao → if (Condicao) then Comando ComandoElse

ComandoElse → else Comando
| ε

ComandoRepeticao → while(Condicao) do Comando
| repeat Comando until (Condicao) ;

Condicao → Expressao OperadorRelacional Expressao

OperadorRelacional → ==
| !=

| <
| >
| <=
| >=

Expressao → Termo Expressao'

Expressao' → + Termo Expressao'
| - Termo Expressao'
| ε

Termo → Fator Termo'

Termo' → * Fator Termo'
| / Fator Termo'
| ** Fator Termo'
| ε

Fator → (Expressao)
| ID
| INT_CONST
| FLOAT_CONST
| CHAR_CONST

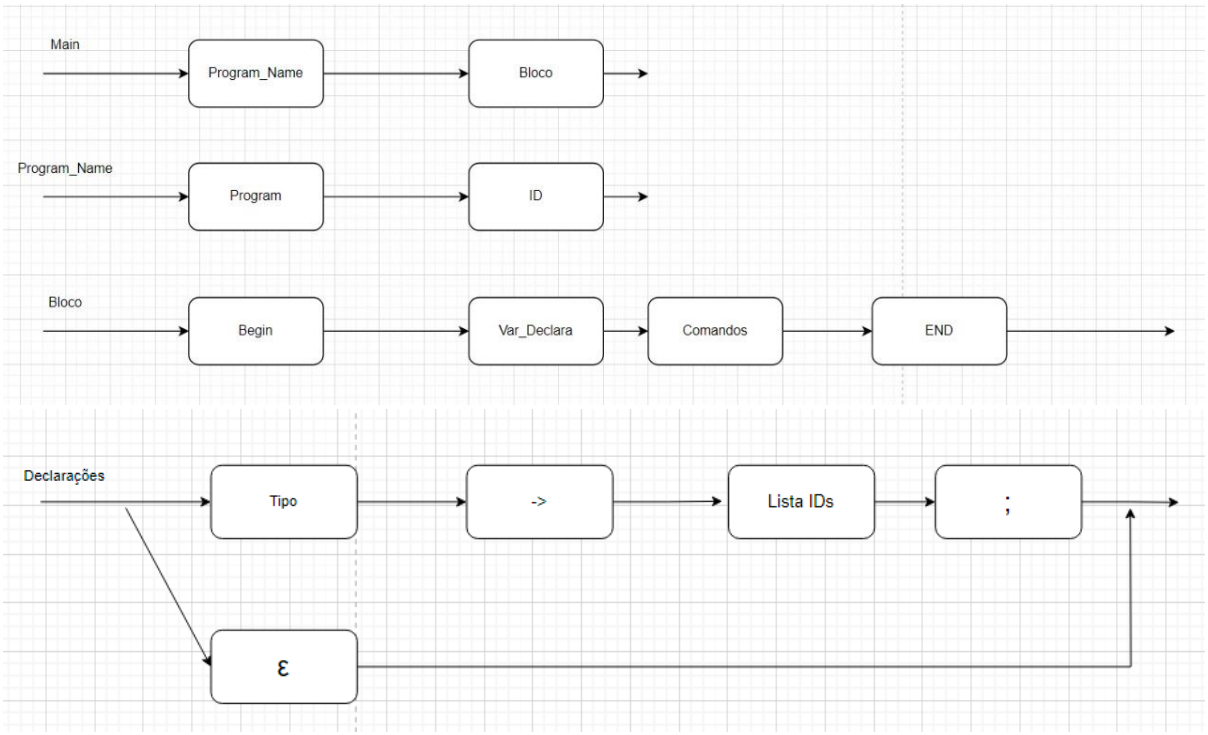
– Cálculo de FIRST e FOLLOW para os símbolos da gramática

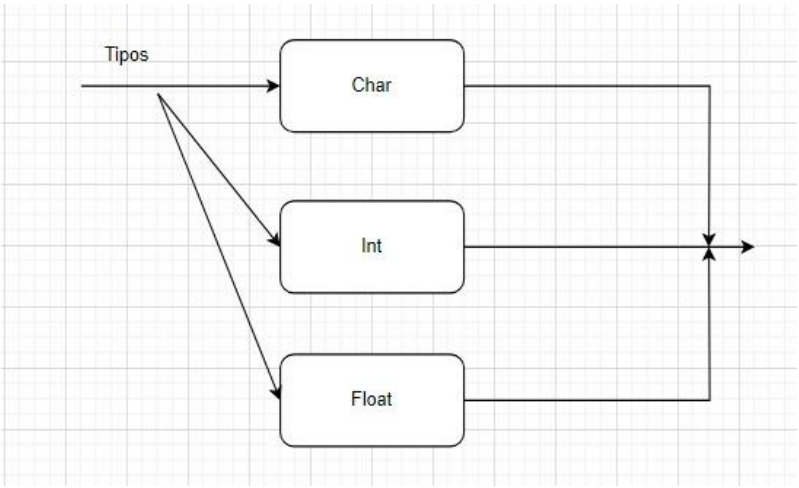
- **Conjunto FIRST**

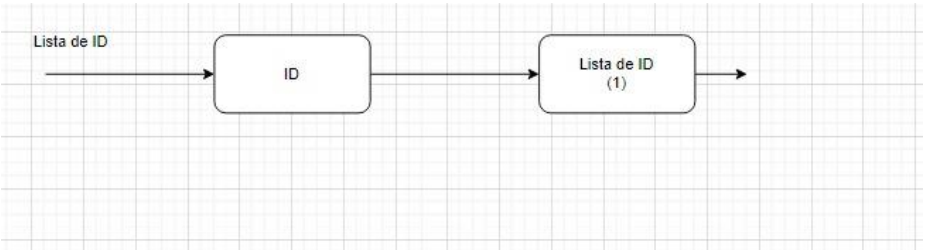
  - FIRST(Programa) = {main}
  - FIRST(Bloco) = {begin}
  - FIRST(Declaracoes) = {int, char, float, ε}
  - FIRST(DeclaracaoVar) = {int, char, float}
  - FIRST(Tipo) = {int, char, float}
  - FIRST(ListaIds) = {tipo}
  - FIRST(ListaIds') = {,, ε}
  - FIRST(Comandos) = {ID, if, while, repeat, begin, {Comentario}, ε}
  - FIRST(Comando) = {ID, if, while, repeat, begin, {Comentario}}
  - FIRST(Atribuicao) = {ID}
  - FIRST(ComandoSelecao) = {if}
  - FIRST(ComandoElse) = {else, ε}
  - FIRST(ComandoRepeticao) = {while, repeat}
  - FIRST(Condicao) = {ID, INT_CONST, FLOAT_CONST, CHAR_CONST, (}
  - FIRST(OperadorRelacional) = {==, !=, <, >, <=, >=}
  - FIRST(Expressao) = {ID, INT_CONST, FLOAT_CONST, CHAR_CONST, (}
  - FIRST(Expressao') = {+, -, ε}
  - FIRST(Termo) = {ID, INT_CONST, FLOAT_CONST, CHAR_CONST, (}
  - FIRST(Termo') = {*, /, , ε}
  - FIRST(Fator) = {ID, INT_CONST, FLOAT_CONST, CHAR_CONST, (}

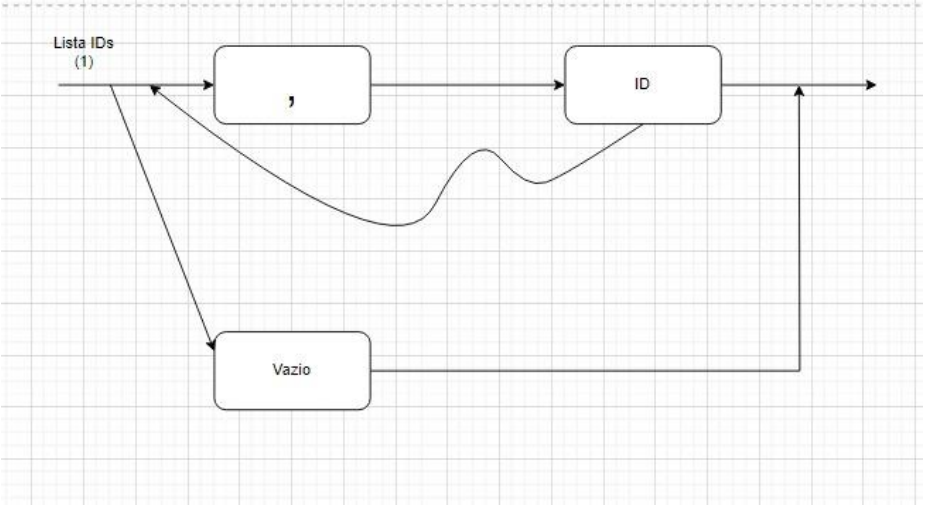
- **Conjunto FOLLOW**

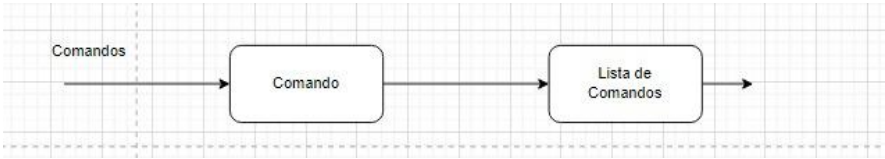
  - FOLLOW(Programa) = {EOF}
  - FOLLOW(Bloco) = {EOF}
  - FOLLOW(Declaracoes) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(DeclaracaoVar) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(Tipo) = {:}
  - FOLLOW(ListaIds) = {;}
  - FOLLOW(ListaIds') = {;}
  - FOLLOW(Comandos) = {end}
  - FOLLOW(Comando) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(Atribuicao) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(ComandoSelecao) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(ComandoElse) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(ComandoRepeticao) = {ID, if, while, repeat, begin, end, {Comentario}}
  - FOLLOW(Condicao) = {)}
  - FOLLOW(OperadorRelacional) = {ID, INT_CONST, FLOAT_CONST, CHAR_CONST, (}
  - FOLLOW(Expressao) = {==, !=, <, >, <=, >=,), ;}
  - FOLLOW(Expressao') = {==, !=, <, >, <=, >=,), ;}
  - FOLLOW(Termo) = {+, -, ==, !=, <, >, <=, >=,), ;}
  - FOLLOW(Termo') = {+, -, ==, !=, <, >, <=, >=,), ;}
  - FOLLOW(Fator) = {*, /, , +, -, ==, !=, <, >, <=, >=,), ;}

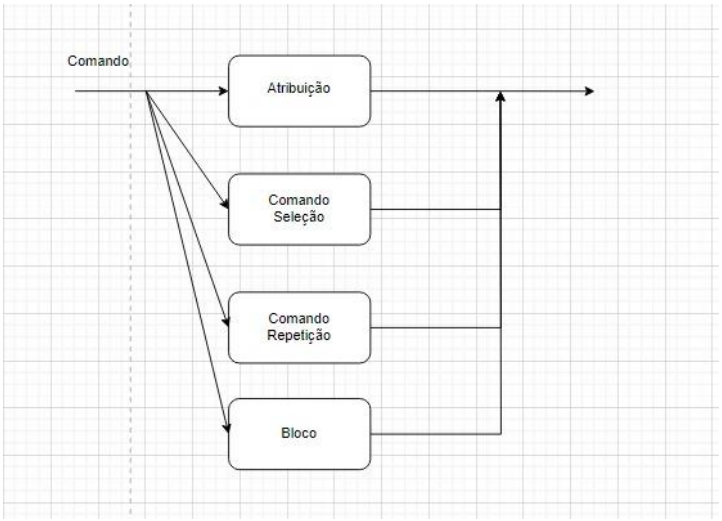


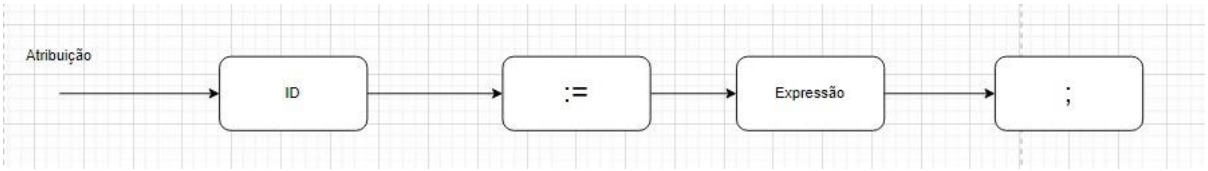

Tipos
Char
Int
Float

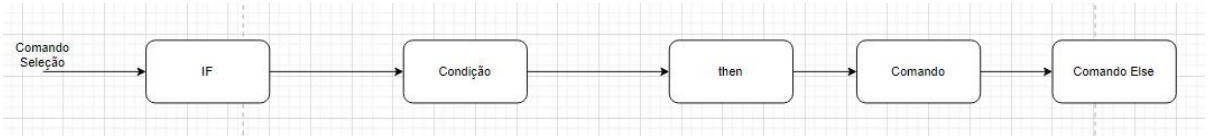

Lista de ID
ID
Lista de ID
(1)

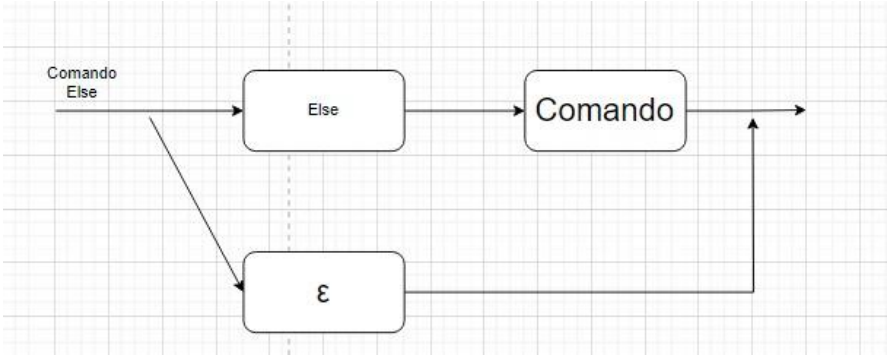

Lista IDs
(1)
,
ID
Vazio

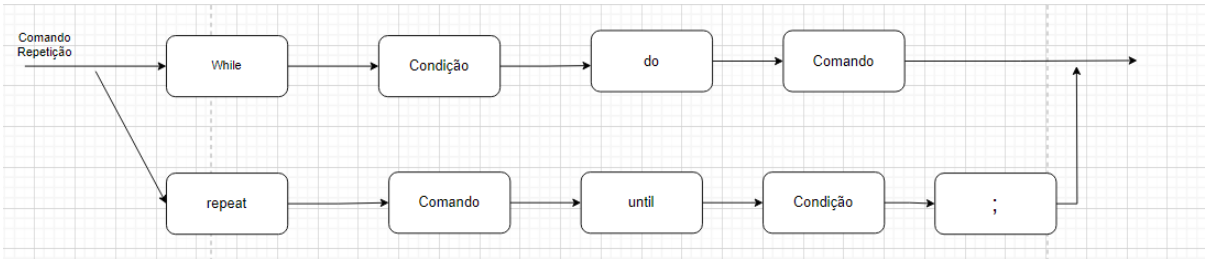

Comandos
Comando
Lista de
Comandos

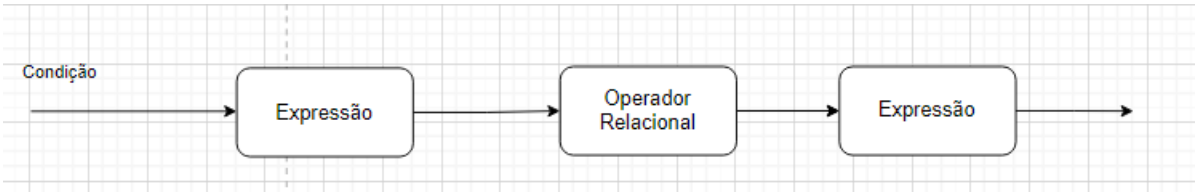

Comando
Atribuição
Comando Seleção
Comando Repetição
Bloco

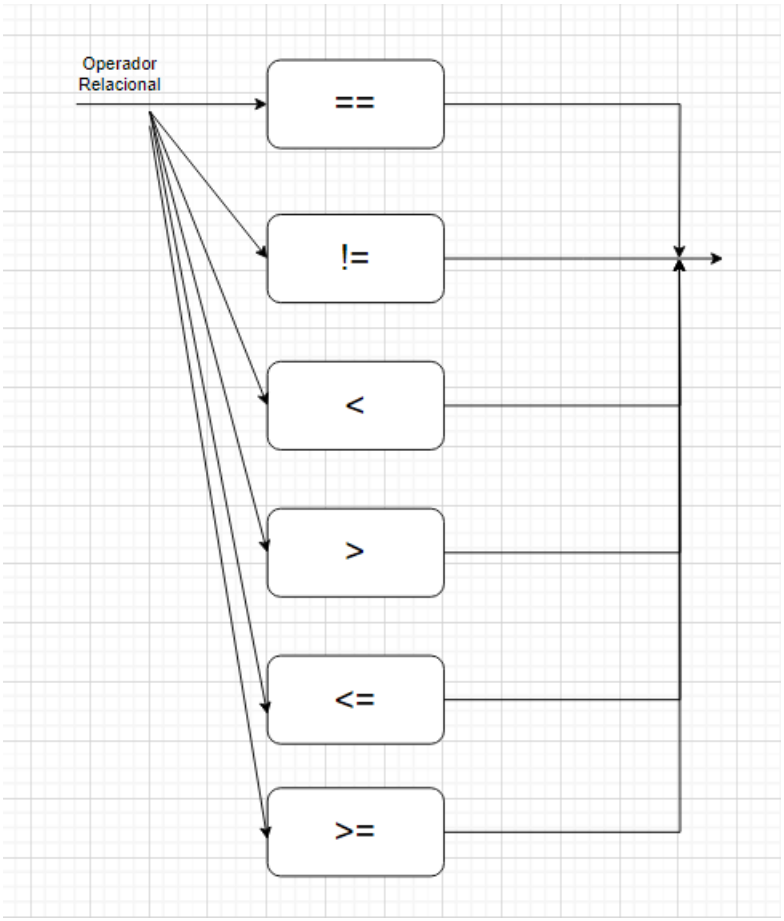

Atribuição
ID
:=
Expressão
;

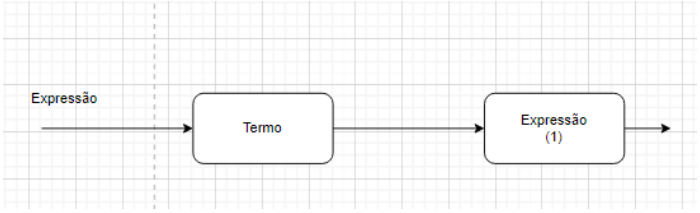

Comando Seleção
IF
Condição
then
Comando
Comando Else

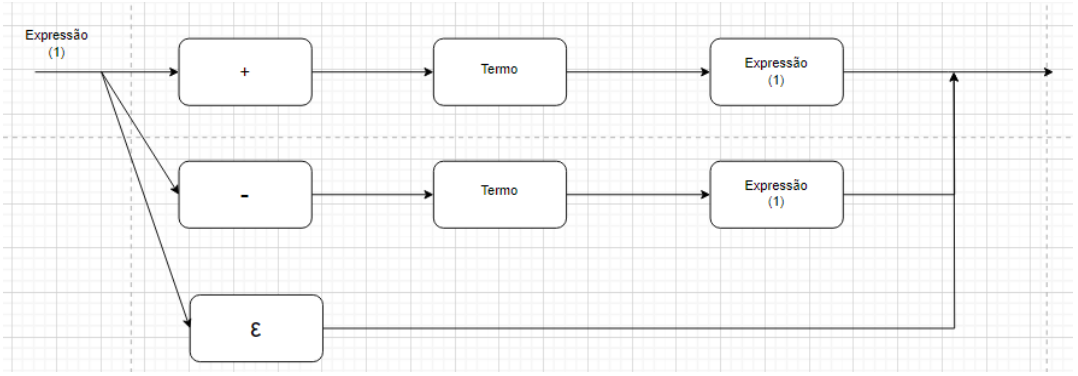

Comando Else
Else
Comando
ε

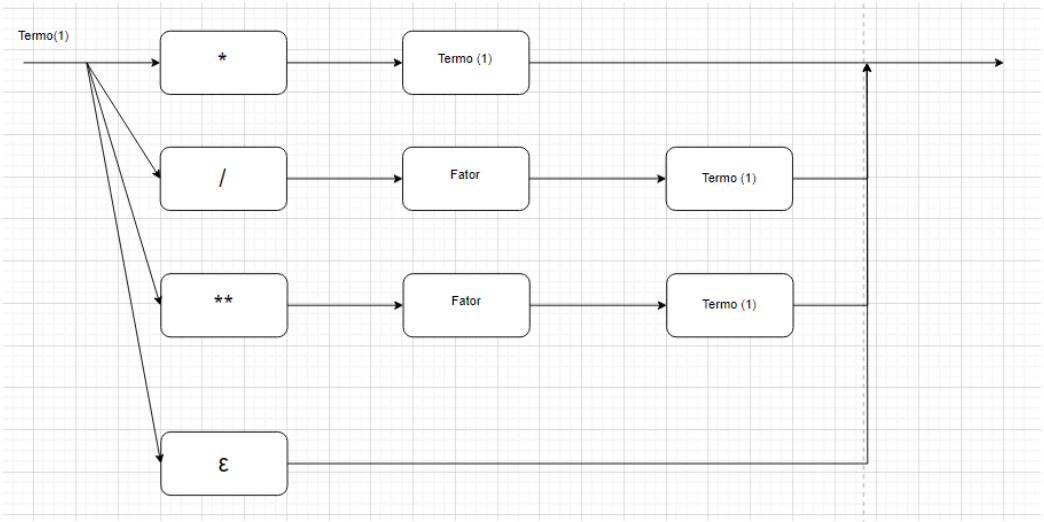

Comando Repetição
While
Condição
do
Comando
repeat
Comando
until
Condição
;

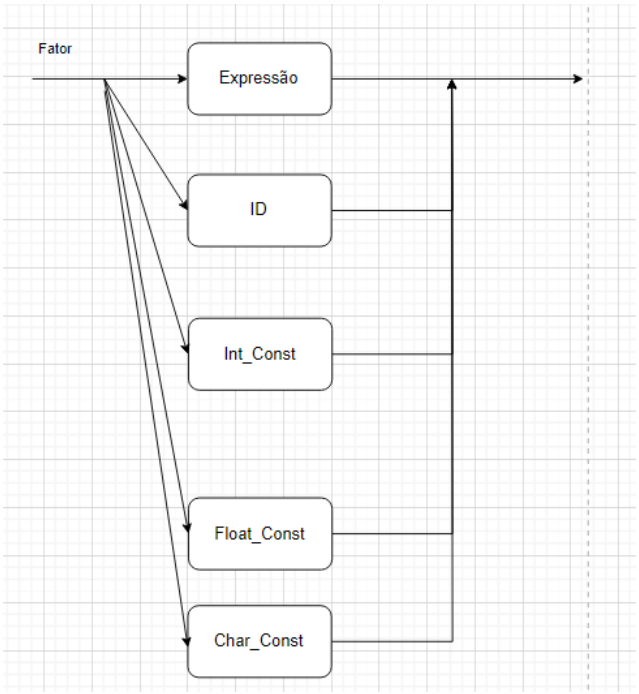

Condição
Expressão
Operador Relacional
Expressão


Operador
Relacional
==
!=
<
>
<=
>=


Expressão
Termo
Expressão
(1)


Expressão
(1)
+
Termo
Expressão
(1)
-
Termo
Expressão
(1)
ε


Termo(1)
*
Termo (1)
/
Fator
Termo (1)
**
Fator
Termo (1)
ε


Fator
Expressão
ID
Int_Const
Float_Const
Char_Const