

# Projeto de Compiladores

## Analizador Léxico e Sintático da linguagem hipotética Algox

### 1. Introdução

Este documento descreve a linguagem *Algox*, uma linguagem de algoritmo hipotética, que será utilizada como exemplo na demonstração dos conceitos envolvidos na introdução a compiladores. A linguagem *Algox* é simples e permite descrever algoritmos básicos envolvendo operações aritméticas e comandos de entrada e saída em um console.

### 2. Estrutura geral de um programa em Algox

```
PROGRAMA <nome_do_programa>
INICIO
    <corpo_do_programa>
FIM
```

O corpo do programa é dividido em duas seções. (Trechos entre chaves {} são comentários em linha).

```
{ Área de declarações }
<declaracoes>
{ Algoritmo }
<algoritmo>
```

#### 2.1 Estrutura das Declarações

As declarações seguem o seguinte formato TIPO NOME onde NOME é uma sequência de letras e números iniciado com uma letra, e TIPO é “INTEIRO”, “REAL”, “CHARACTER”, “CADEIA”, “LISTA\_INT” ou “LISTA\_REAL”.

INTEIRO var1	{ número inteiro }
REAL var2	{ número real }
CHARACTER A, B	{ um caracter }
CADEIA C[30], D[10]	{ sequência de símbolos }
LISTA_INT vetor[10]	{ arranjo de inteiros }
LISTA_REAL VetReal[50]	{ arranjo de reais }

#### 2.2 Estrutura do Algoritmo

O algoritmo consiste em uma sequência de comandos.

### 3. Comandos

Os comandos na linguagem possibilitam ações de atribuição, entrada, saída, seleção e repetição criadas pelo programador.

#### 3.1 Comando de Atribuição

O comando de atribuição (notação “:=”), armazena um valor em uma variável, seguindo o formato VARIÁVEL := EXPRESSÃO

```
var1 := 1000           { atribuição direta }
var3 := var1 + var2    { atribuição do resultado da operação }
```

#### 3.2 Comando de Entrada

O comando de entrada (notação “LEIA”), faz a leitura do usuário e armazena o valor lido em uma variável, seguindo o formato LEIA VARIÁVEL ou LEIA VARIÁVEL 1, VARIÁVEL 2, ...

```
LEIA var1
LEIA var3
LEIA var1, var2, var3
```

#### 3.3 Comando de Saída

O comando de saída (notação “ESCREVA”), imprime o valor de uma variável ou uma constante do tipo cadeia de caracteres no console, seguindo o formato ESCREVA VARIÁVEL ou ESCREVA CADEIA ou ESCREVA CADEIA, VARIÁVEL ou ESCREVA VARIÁVEL, CADEIA.

```
{ Tudo entre aspas simples (') é considerado parte de uma cadeia }

ESCREVA var1
ESCREVA var3
ESCREVA 'Alô Mundo'
ESCREVA 'Resto da divisão =', resto
ESCREVA cm, 'cm'
```

### 3.4 Comando de Seleção

O comando de seleção (notação “SE”) permite especificar um desvio condicional de fluxo e segue o seguinte formato

```
SE <expressao_relacional>  
    ENTAO <comando>  
FIM_SE
```

Onde <expressao\_relacional> se refere a uma expressão relacional e <comando> a um ou mais comandos. Por exemplo:

```
SE var1 .M. var2  
    ENTAO ESCREVA var1  
FIM_SE  
  
SE var2 .M. var3  
    ENTAO var1 := var2 + var3  
        ESCREVA 'soma =', var1  
FIM_SE
```

### 3.4 Comando de Repetição

O comando de repetição (notação “ENQUANTO”) permite que um determinado comando seja repetido conforme alguma condição. Segue um formato similar ao do comando de seleção:

```
ENQUANTO <expressao_relacional>  
    <comando>  
FIM_ENQUANTO  
  
ENQUANTO var1 .M. 10  
    var1 := var1 - 1  
FIM_ENQUANTO
```

## 4. Expressões

Os comandos podem fazer uso de expressões. Existem três tipos de expressões: expressões aritméticas, expressões relacionais e expressões lógicas

### 4.1 Expressões Aritméticas

As expressões aritméticas seguem as regras convencionais de precedência e associatividade: operadores “\*” e “/” tem precedência sobre “+” e “-”, e o parêntesis pode ser usado para forçar a precedência.

1000	{ constante inteira }
3,14	{ constante real }
var1	{ variável }
var1 + 5	{ operação aritmética }
2+3*5	{ operações aritméticas compostas }
(2 + 3) * 5	{ operações aritméticas compostas }

### 4.2 Expressões Relacionais

Uma expressão relacional envolve apenas dois operadores relacionais. Esses operadores têm a mesma precedência e são associativos à esquerda. Além disso, só podem ser utilizados entre duas expressões aritméticas.

{ menor ou igual (<=) anotado como ".M." }	
{ igual (=) anotado como ".I." }	
var1 .M. var2	{ comparação entre duas variáveis }
var1 * var2 .M. var3	{ comparação entre operação e variável }

### 4.3 Expressões Lógicas

Os operadores lógicos “ou” e “e” podem ser utilizados para criação de expressões relacionais compostas, e o operador lógico de negação para criação de expressões relacionais simples. Os operadores relacionais possuem precedência sobre os lógicos e o uso do parênteses pode ser usado para forçar a precedência entre dois operadores lógicos.

{ operador "e" anotado como ".E." }	
{ operador "ou" anotado como ".OU." }	
{ operador de negação anotado como ".N." }	
var1 .M. var2 .E. var1 .M. var3	{ expressão relacional composta }
var1 .M. var2 .OU. var1 .M. var3	{ expressão relacional composta }
.N. (var1 .I. var2)	{ expressão relacional simples }

## 5. Exemplos de Programas em Algol

```
PROGRAMA fatorial_exemplo
INICIO
  { DECLARAÇÕES }
  INTEIRO argumento, fatorial
  { ALGORITMO }
  { Calcula o fatorial de um número inteiro }
  LEIA argumento
  fatorial := argumento
  SE argumento .I. 0
    ENTAO fatorial := 1
  FIM_SE
  ENQUANTO argumento .M. 1
    fatorial := fatorial * argumento
    argumento := argumento - 1
  FIM_ENQUANTO
  ESCRIVA 'fatorial =', fatorial
FIM
```

```
PROGRAMA leitura de lista
INICIO
  { DECLARAÇÕES }
  INTEIRO n, i, x, k
  LISTA_INT A[100]
  { ALGORITMO }
  { Armazena os dados da lista }
  ESCRIVA 'quantos números vai armazenar?'
  LEIA n
  i := 1
  ENQUANTO i .M. n
    LEIA A[i]
    i := i + 1
  FIM_ENQUANTO
  { Escreve a lista de números }
  ESCRIVA 'Número armazenados:'
  i := 1
  ENQUANTO i .M. n
    ESCRIVA A[i], ' '
    i := i + 1
  FIM_ENQUANTO
FIM
```

## 6. Gramática da linguagem Algox

Na gramática a seguir os não-terminais são representados entre  $\langle \rangle$  e em *snake\_case*, enquanto os terminais são representados pelos tokens equivalentes gerados pelo analisador léxico. A expressão regular referente aos tokens está descrita na Seção 7 sobre o analisador léxico.

```
GALGOX = ( { <alcox>, <nome_do_programa>, <nome_do_programa_aux>,
<corpo_do_programa>, <declaracoes>, <declaracoes_aux>, <declaracao>,
<declaracao_aux>, <algoritmo>, <algoritmo_aux>, <comando>, <comando_atribuicao>,
<comando_entrada>, <comando_entrada_aux>, <comando_saida>, <comando_saida_aux>,
<comando_saida_variavel_cadeia>, <comando_saida_cadeia_variavel>,
<comando_selecao>, <comando_repeticao>, <expressao_aritmetica>,
<expressao_aritmetica_aux>, <expressao_relacional>, <expressao_logica>,
<expressao_logica_aux>, <constante>, <variavel>, <operador_aritmetico>,
<operador_relacional>, <operador_logico>, <tipo> }, { ABRE, ENQUANTO, ENTAO,
ESCREVA, FECHA, FIM, FIM_ENQUANTO, FIM_SE, INICIO, LEIA, OPERADOR_ADICAO,
OPERADOR_AND, OPERADOR_ATRIBUICAO, OPERADOR_DIVISAO, OPERADOR_IGUAL,
OPERADOR_MENOR_IGUAL, OPERADOR_NOT, OPERADOR_OR, OPERADOR_PRODUTO,
OPERADOR_SUBTRACAO, PROGRAMA, SE, TIPO_CADEIA, TIPO_CARACTER, TIPO_INTEIRO,
TIPO_LISTA_INT, TIPO_LISTA_REAL, TIPO_REAL, VIRGULA, DIGITO, CONSTANTE_INTEIRA,
CONSTANTE_REAL, LETRA, PALAVRA, VARIABEL, VARIABEL_LISTA, CADEIA, COMENTARIO },
PALGOX, <alcox> )
PALGOX = {
  <alcox> → PROGRAMA <nome_do_programa> INICIO <corpo_do_programa> FIM
  <nome_do_programa> → VARIABEL <nome_do_programa_aux>
  <nome_do_programa_aux> → VARIABEL <nome_do_programa> | ε
  <corpo_do_programa> → <declaracoes> <algoritmo>
  <declaracoes> → <declaracao> <declaracoes_aux>
  <declaracoes_aux> → <declaracao> <declaracoes_aux> | ε
  <declaracao> → <tipo> <variavel> <declaracao_aux>
  <declaracao_aux> → VIRGULA <variavel> <declaracao_aux> | ε
  <algoritmo> → <comando> <algoritmo_aux>
  <algoritmo_aux> → <comando> <algoritmo_aux> | ε
  <comando> → <comando_atribuicao> | <comando_entrada> | <comando_saida> |
<comando_selecao> | <comando_repeticao>
  <comando_atribuicao> → <variavel> OPERADOR_ATRIBUICAO <expressao_aritmetica>
  <comando_entrada> → LEIA <variavel> <comando_entrada_aux>
  <comando_entrada_aux> → VIRGULA <variavel> | ε
  <comando_saida> → ESCREVA <comando_saida_aux>
  <comando_saida_aux> → <variavel> <comando_saida_variavel_cadeia> | CADEIA
<comando_saida_cadeia_variavel>
  <comando_saida_variavel_cadeia> → VIRGULA CADEIA | ε
  <comando_saida_cadeia_variavel> → VIRGULA <variavel> | ε
  <comando_selecao> → SE <expressao_relacional> ENTAO <comando> FIM_SE
```

```

<comando_repeticao> → ENQUANTO <expressao_relacional> <comando> FIM_ENQUANTO
<expressao_aritmetica> → <constante> <expressao_aritmetica_aux> | <variavel>
<expressao_aritmetica_aux> | ABRE <expressao_aritmetica> FECHA
<expressao_aritmetica_aux>
<expressao_aritmetica_aux> → <operador_aritmetico> <expressao_aritmetica> | ε
<expressao_relacional> → <expressao_aritmetica> <operador_relacional>
<expressao_aritmetica>
<expressao_logica> → <expressao_relacional> <expressao_logica_aux> |
OPERADOR_NOT ABRE <expressao_logica> FECHA <expressao_logica_aux> | ABRE
<expressao_logica> FECHA <expressao_logica_aux>
<expressao_logica_aux> → <operador_logico> <expressao_logica> | ε
<constante> → CONSTANTE_INTEIRA | CONSTANTE_REAL
<variavel> → VARIABEL | VARIABEL_LISTA
<operador_aritmetico> → OPERADOR_ADICAO | OPERADOR_SUBTRACAO | OPERADOR_PRODUTO
| OPERADOR_DIVISAO
<operador_relacional> → OPERADOR_MENOR_IGUAL | OPERADOR_IGUAL
<operador_logico> → OPERADOR_AND | OPERADOR_OR
<tipo> → TIPO_INTEIRO | TIPO_REAL | TIPO_CARACTER | TIPO_CADEIA | TIPO_LISTA_INT
| TIPO_LISTA_REAL
}

```

## 7. Analisador Léxico (FLEX)

O analisador léxico foi montado em FLEX e o código está disponível no [repositório do projeto](#) no GitHub.

### 7.1 Expressões Regulares dos Tokens

A seguir, estão representados as expressões regulares utilizadas no analisador léxico. Os tokens usados na gramática estão representados aqui.

ABRE	"("
ENQUANTO	"ENQUANTO"
ENTAO	"ENTAO"
ESCREVA	"ESCREVA"
FECHA	")"
FIM	"FIM"
FIM_ENQUANTO	"FIM_ENQUANTO"
FIM_SE	"FIM_SE"
INICIO	"INICIO"
LEIA	"LEIA"
OPERADOR_ADICAO	"+"
OPERADOR_AND	".E."
OPERADOR_ATRIBUICAO	";="

OPERADOR_DIVISAO	"\\"
OPERADOR_IGUAL	".I."
OPERADOR_MENOR_IGUAL	".M."
OPERADOR_NOT	".N."
OPERADOR_OR	".OU."
OPERADOR_PRODUTO	"*"
OPERADOR_SUBTRACAO	"_"
PROGRAMA	"PROGRAMA"
SE	"SE"
TIPO_CADEIA	"CADEIA"
TIPO_CARACTER	"CARACTER"
TIPO_INTEIRO	"INTEIRO"
TIPO_LISTA_INT	"LISTA_INT"
TIPO_LISTA_REAL	"LISTA_REAL"
TIPO_REAL	"REAL"
VIRGULA	","
DIGITO	([0-9])
CONSTANTE_INTEIRA	({DIGITO}+)
CONSTANTE_REAL	({DIGITO}+", "%{DIGITO}+)
LETRA	([a-zA-Z])
PALAVRA	([a-zA-Z_-])
VARIAVEL	({LETRA}({PALAVRA} {CONSTANTE_INTEIRA})*)
VARIAVEL_LISTA	({VARIAVEL}"("({VARIAVEL} {CONSTANTE_INTEIRA})")")
CADEIA	(\[^\r\n]+\')
COMENTARIO	(\[.*\])

## 7.2 Regras do Analisador Léxico

Abaixo as regras para preenchimento da lista de tokens identificados na entrada.

{ABRE}	{ printf("ABRE\n"); }
{ENQUANTO}	{ printf("ENQUANTO\n"); }
{ENTAO}	{ printf("ENTAO\n"); }
{ESCREVA}	{ printf("ESCREVA\n"); }
{FECHA}	{ printf("FECHA\n"); }
{FIM_ENQUANTO}	{ printf("FIM_ENQUANTO\n"); }
{FIM_SE}	{ printf("FIM_SE\n"); }
{FIM}	{ printf("FIM\n"); }
{INICIO}	{ printf("INICIO\n"); }
{LEIA}	{ printf("LEIA\n"); }
{OPERADOR_ADICAO}	{ printf("OPERADOR_ADICAO\n"); }
{OPERADOR_AND}	{ printf("OPERADOR_AND\n"); }
{OPERADOR_ATRIBUICAO}	{ printf("OPERADOR_ATRIBUICAO\n"); }
{OPERADOR_DIVISAO}	{ printf("OPERADOR_DIVISAO\n"); }



```

{OPERADOR_IGUAL}      { printf("OPERADOR_IGUAL\n"); }
{OPERADOR_MENOR_IGUAL} { printf("OPERADOR_MENOR_IGUAL\n"); }
{OPERADOR_NOT}        { printf("OPERADOR_NOT\n"); }
{OPERADOR_OR}         { printf("OPERADOR_OR\n"); }
{OPERADOR_PRODUTO}    { printf("OPERADOR_PRODUTO\n"); }
{OPERADOR_SUBTRACAO}  { printf("OPERADOR_SUBTRACAO\n"); }
{PROGRAMA}            { printf("PROGRAMA\n"); }
{SE}                  { printf("SE\n"); }
{TIPO_CADEIA}         { printf("TIPO_CADEIA\n"); }
{TIPO_CARACTER}       { printf("TIPO_CARACTER\n"); }
{TIPO_INTEIRO}        { printf("TIPO_INTEIRO\n"); }
{TIPO_LISTA_INT}      { printf("TIPO_LISTA_INT\n"); }
{TIPO_LISTA_REAL}     { printf("TIPO_LISTA_REAL\n"); }
{TIPO_REAL}           { printf("TIPO_REAL\n"); }
{VIRGULA}             { printf("VIRGULA\n"); }
{CONSTANTE_INTEIRA}   { printf("CONSTANTE_INTEIRA\n"); }
{CONSTANTE_REAL}      { printf("CONSTANTE_REAL\n"); }
{VARIAVEL}            { printf("VARIAVEL\n"); }
{VARIAVEL_LISTA}      { printf("VARIAVEL_LISTA\n"); }
{CADEIA}              { printf("CADEIA\n"); }
{COMENTARIO}          // Pula os trechos comentados
[ \t\r\n]+           // Ignora espaços em branco, tabulação e quebras de linha
.                    { printf("ERRO\n"); } // Entrada não reconhecida
<<EOF>>             { printf("$"); return 0; } // $ indica o final da entrada

```

### 7.3 Resultados

Aplicando os exemplos "fatorial\_exemplo" e "leitura de lista" da seção 5 no analisador sintático obtemos as seguintes listas de tokens.

leitura de lista	fatorial_exemplo
PROGRAMA	PROGRAMA
VARIAVEL	VARIAVEL
VARIAVEL	INICIO
VARIAVEL	TIPO_INTEIRO
INICIO	VARIAVEL
TIPO_INTEIRO	VIRGULA
VARIAVEL	VARIAVEL
VIRGULA	LEIA
VARIAVEL	VARIAVEL
VIRGULA	VARIAVEL
VARIAVEL	OPERADOR_ATRIBUICAO

VIRGULA	VARIAVEL
VARIAVEL	SE
TIPO_LISTA_INT	VARIAVEL
VARIAVEL_LISTA	OPERADOR_IGUAL
ESCREVA	CONSTANTE_INTEIRA
CADEIA	ENTAO
LEIA	VARIAVEL
VARIAVEL	OPERADOR_ATRIBUICAO
VARIAVEL	CONSTANTE_INTEIRA
OPERADOR_ATRIBUICAO	FIM_SE
CONSTANTE_INTEIRA	ENQUANTO
ENQUANTO	VARIAVEL
VARIAVEL	OPERADOR_MENOR_IGUAL
OPERADOR_MENOR_IGUAL	CONSTANTE_INTEIRA
VARIAVEL	VARIAVEL
LEIA	OPERADOR_ATRIBUICAO
VARIAVEL_LISTA	VARIAVEL
VARIAVEL	OPERADOR_PRODUTO
OPERADOR_ATRIBUICAO	VARIAVEL
VARIAVEL	VARIAVEL
OPERADOR_ADICAO	OPERADOR_ATRIBUICAO
CONSTANTE_INTEIRA	VARIAVEL
FIM_ENQUANTO	OPERADOR_SUBTRACAO
ESCREVA	CONSTANTE_INTEIRA
CADEIA	FIM_ENQUANTO
VARIAVEL	ESCREVA
OPERADOR_ATRIBUICAO	CADEIA
CONSTANTE_INTEIRA	VIRGULA
ENQUANTO	VARIAVEL
VARIAVEL	FIM
OPERADOR_MENOR_IGUAL	\$
VARIAVEL	
ESCREVA	
VARIAVEL_LISTA	
VIRGULA	
CADEIA	
VARIAVEL	
OPERADOR_ATRIBUICAO	
VARIAVEL	
OPERADOR_ADICAO	
CONSTANTE_INTEIRA	
FIM_ENQUANTO	
FIM	
\$	

## 8. Analisador Sintático

O analisador sintático foi escrito em linguagem C e está disponível no [repositório do projeto](#) no GitHub.

### 8.1 Primeiros e Seguidores da Gramática

#### 8.1.1 Primeiros

```
Prim(<algor>) = { PROGRAMA }
Prim(<nome_do_programa>) = { VARIABEL }
Prim(<nome_do_programa_aux>) = { VARIABEL, ε }
Prim(<corpo_do_programa>) = { TIPO_INTEIRO, TIPO_REAL,
TIPO_CHARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracoes>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CHARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracoes_aux>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CHARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Prim(<declaracao>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CHARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracao_aux>) = { VIRGULA, ε }
Prim(<algoritmo>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA, SE,
ENQUANTO }
Prim(<algoritmo_aux>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA,
SE, ENQUANTO, ε }
Prim(<comando>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA, SE,
ENQUANTO }
Prim(<comando_atribuicao>) = { VARIABEL, VARIABEL_LISTA }
Prim(<comando_entrada>) = { LEIA }
Prim(<comando_entrada_aux>) = { VIRGULA, ε }
Prim(<comando_saida>) = { ESCREVA }
Prim(<comando_saida_aux>) = { VARIABEL, VARIABEL_LISTA, CADEIA }
Prim(<comando_saida_variavel_cadeia>) = { VIRGULA, ε }
Prim(<comando_saida_cadeia_variavel>) = { VIRGULA, ε }
Prim(<comando_selecao>) = { SE }
Prim(<comando_repeticao>) = { ENQUANTO }
Prim(<expressao_aritmetica>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE }
Prim(<expressao_aritmetica_aux>) = { OPERADOR_ADICAO,
OPERADOR_SUBTRACAO, OPERADOR_PRODUTO, OPERADOR_DIVISAO, ε }
Prim(<expressao_relacional>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE }
Prim(<expressao_logica>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE, OPERADOR_NOT }
Prim(<expressao_logica_aux>) = { OPERADOR_AND, OPERADOR_OR, ε }
```

```

Prim(<constante>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL }
Prim(<variavel>) = { VARIABEL, VARIABEL_LISTA }
Prim(<operador_aritmetico>) = { OPERADOR_ADICAO, OPERADOR_SUBTRACAO,
OPERADOR_PRODUTO, OPERADOR_DIVISAO }
Prim(<operador_relacional>) = { OPERADOR_MENOR_IGUAL, OPERADOR_IGUAL
}
Prim(<operador_logico>) = { OPERADOR_AND, OPERADOR_OR }
Prim(<tipo>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }

```

### 8.1.2 Seguidores

```

Seg(<algoo>) = { $ }
Seg(<nome_do_programa>) = { INICIO }
Seg(<nome_do_programa_aux>) = { INICIO }
Seg(<corpo_do_programa>) = { FIM }
Seg(<declaracoes>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA, SE,
ENQUANTO }
Seg(<declaracoes_aux>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, VARIABEL,
VARIABEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, ε }
Seg(<declaracao>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Seg(<declaracao_aux>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER,
TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Seg(<algoritmo>) = { FIM }
Seg(<algoritmo_aux>) = { FIM }
Seg(<comando>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA, SE,
ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_atribuicao>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_entrada>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA,
SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_entrada_aux>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA,
SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_aux>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_variavel_cadeia>) = { VARIABEL, VARIABEL_LISTA,
LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_cadeia_variavel>) = { VARIABEL, VARIABEL_LISTA,
LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_selecao>) = { VARIABEL, VARIABEL_LISTA, LEIA, ESCREVA,

```

```

SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_repeticao>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<expressao_aritmetica>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, FECHA,
OPERADOR_MENOR_IGUAL, OPERADOR_IGUAL, ENTAO, OPERADOR_AND,
OPERADOR_OR, ε }
Seg(<expressao_aritmetica_aux>) = { VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, FECHA,
OPERADOR_MENOR_IGUAL, OPERADOR_IGUAL, ENTAO, OPERADOR_AND,
OPERADOR_OR, ε }
Seg(<expressao_relacional>) = { ENTAO, VARIABEL, VARIABEL_LISTA,
LEIA, ESCREVA, SE, ENQUANTO, OPERADOR_AND, OPERADOR_OR, ε }
Seg(<expressao_logica>) = { FECHA }
Seg(<expressao_logica_aux>) = { FECHA }
Seg(<constante>) = { OPERADOR_ADICAO, OPERADOR_SUBTRACAO,
OPERADOR_PRODUTO, OPERADOR_DIVISAO, ε }
Seg(<variavel>) = { VIRGULA, VARIABEL, VARIABEL_LISTA, LEIA,
ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, OPERADOR_ATRIBUICAO,
OPERADOR_ADICAO, OPERADOR_SUBTRACAO, OPERADOR_PRODUTO,
OPERADOR_DIVISAO, ε }
Seg(<operador_aritmetico>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE }
Seg(<operador_relacional>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE }
Seg(<operador_logico>) => { CONSTANTE_INTEIRA, CONSTANTE_REAL,
VARIABEL, VARIABEL_LISTA, ABRE, OPERADOR_NOT }
Seg(<tipo>) = { VARIABEL, VARIABEL_LISTA }

```

## 8.2 Tabela LL(1) da Gramática Algox

A tabela pode ser acessada no [repositório do projeto](#) no GitHub. Ela precisou ser criada em uma planilha devido a grande quantidade de não-terminais e terminais da gramática.

## 8.3 Pseudocódigo

Abaixo o pseudocódigo relativo a gramática Algox.

```

input ← vetor de strings
indice ← 0

se analisarAlgox()
    retorna RECONHECE
senão

```

retorna NÃO RECONHECE

```
função analisarPalavraReservada(palavra: string) -> boolean
  se input[índice] seja igual a palavra
    índice++
    retornar verdadeiro
  retornar falso
```

```
função analisarAlgox() -> boolean
  se analisarPalavraReservada("PROGRAMA")
    se analisarNomePrograma()
      se analisarPalavraReservada("INICIO")
        se analisarCorpoPrograma()
          se analisarPalavraReservada("FIM")
            retornar verdadeiro
        retornar falso
    retornar falso
```

```
função analisarNomePrograma() -> boolean
  se analisarPalavraReservada("VARIABEL")
    se analisarNomeProgramaAux()
      retornar verdadeiro
    retornar falso
```

```
função analisarNomeProgramaAux() -> boolean
  se analisarPalavraReservada("VARIABEL")
    se analisarNomePrograma()
      retornar verdadeiro
  senão
    retornar verdadeiro // ε (vazio)
```

```
função analisarCorpoPrograma() -> boolean
  se analisarDeclaracoes()
    se analisarAlgoritmo()
      retornar verdadeiro
  retornar falso
```

```
função analisarDeclaracoes() -> boolean
  se analisarDeclaracao()
    se analisarDeclaracoesAux()
      retornar verdadeiro
  retornar falso
```

```
função analisarDeclaracoesAux() -> boolean
  se analisarDeclaracao()
```

```

        se analisarDeclaracoesAux()
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarDeclaracao() -> boolean
    se analisarTipo()
        se analisarVariavel()
            se analisarDeclaracaoAux()
                retornar verdadeiro
    retornar falso

função analisarDeclaracaoAux() -> boolean
    se input[índice] seja VIRGULA
        índice++
        se analisarVariavel()
            se analisarDeclaracaoAux()
                retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarAlgoritmo() -> boolean
    se analisarComando()
        se analisarAlgoritmoAux()
            retornar verdadeiro
    retornar falso

função analisarAlgoritmoAux() -> boolean
    se analisarComando()
        se analisarAlgoritmoAux()
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarComando() -> boolean
    se analisarComandoAtribuicao()
    ou se analisarComandoEntrada()
    ou se analisarComandoSaida()
    ou se analisarComandoSelecao()
    ou se analisarComandoRepeticao()
        retornar verdadeiro
    retornar falso

função analisarComandoAtribuicao() -> boolean

```

```

    se analisarVariavel()
        se input[índice] seja OPERADOR_ATRIBUICAO
            índice++
            se analisarExpressaoAritmetica()
                retornar verdadeiro
            retornar falso
    retornar falso

função analisarComandoEntrada() -> boolean
    se analisarPalavraReservada("LEIA")
        se analisarVariavel()
            se analisarComandoEntradaAux()
                retornar verdadeiro
        retornar falso
    retornar falso

função analisarComandoEntradaAux() -> boolean
    se input[índice] seja VIRGULA
        índice++
        se analisarVariavel()
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarComandoSaida() -> boolean
    se analisarPalavraReservada("ESCREVA")
        se analisarComandoSaidaAux()
            retornar verdadeiro
        retornar falso
    retornar falso

função analisarComandoSaidaAux() -> boolean
    se analisarVariavel()
        ou se analisarComandoSaidaVariavelCadeia()
        ou se analisarPalavraReservada("CADEIA")
            se analisarComandoSaidaCadeiaVariavel()
                retornar verdadeiro
    retornar falso

função analisarComandoSaidaVariavelCadeia() -> boolean
    se input[índice] seja VIRGULA
        se analisarPalavraReservada("CADEIA")
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarComandoSaidaCadeiaVariavel() -> boolean

```



```

    se input[índice] seja VIRGULA
        índice++
        se analisarVariavel()
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarComandoSelecao() -> boolean
    se analisarPalavraReservada("SE")
        se analisarExpressaoRelacional()
            se analisarPalavraReservada("ENTAO")
                se analisarComando()
                    se analisarPalavraReservada("FIM_SE")
                        retornar verdadeiro
    retornar falso

função analisarComandoRepeticao() -> boolean
    se analisarPalavraReservada("ENQUANTO")
        se analisarExpressaoRelacional()
            se analisarComando()
                se analisarPalavraReservada("FIM_ENQUANTO")
                    retornar verdadeiro
    retornar falso

função analisarExpressaoAritmetica() -> boolean
    se analisarConstante()
    ou se analisarVariavel()
    ou se input[índice] seja ABRE
        índice++
        se analisarExpressaoAritmetica()
            se input[índice] seja FECHA
                índice++
                se analisarExpressaoAritmeticaAux()
                    retornar verdadeiro
    retornar falso

função analisarExpressaoAritmeticaAux() -> boolean
    se analisarOperadorAritmetico()
        se analisarExpressaoAritmetica()
            retornar verdadeiro
    senão
        retornar verdadeiro // ε (vazio)

função analisarExpressaoRelacional() -> boolean

```

```
se analisarExpressaoAritmetica()  
  se analisarOperadorRelacional()  
    se analisarExpressaoAritmetica()  
      retornar verdadeiro  
  retornar falso
```

```
função analisarExpressaoLogica() -> boolean  
  se analisarExpressaoRelacional()  
    se analisarExpressaoLogicaAux()  
      retornar verdadeiro  
  senão se input[índice] seja OPERADOR_NOT  
    índice++  
  se input[índice] seja ABRE  
    índice++  
    se analisarExpressaoLogica()  
      se input[índice] seja FECHA  
        índice++  
        se analisarExpressaoLogicaAux()  
          retornar verdadeiro  
  senão se input[índice] seja ABRE  
    índice++  
  se analisarExpressaoLogica()  
    se input[índice] seja FECHA  
      índice++  
      se analisarExpressaoLogicaAux()  
        retornar verdadeiro  
  retornar falso
```

```
função analisarExpressaoLogicaAux() -> boolean  
  se analisarOperadorLogico()  
    se analisarExpressaoLogica()  
      retornar verdadeiro  
  senão  
    retornar verdadeiro // ε (vazio)
```

```
função analisarConstante() -> boolean  
  se input[índice] seja CONSTANTE_INTEIRA  
    índice++  
    retornar verdadeiro  
  ou se input[índice] seja CONSTANTE_REAL  
    índice++  
    retornar verdadeiro  
  retornar falso
```

função analisarVariavel() -> boolean

se input[índice] seja VARIABEL

índice++

retornar verdadeiro

ou se input[índice] seja VARIABEL\_LISTA

índice++

retornar verdadeiro

retornar falso

função analisarOperadorAritmetico() -> boolean

se input[índice] seja OPERADOR\_ADICAO

ou se input[índice] seja OPERADOR\_SUBTRACAO

ou se input[índice] seja OPERADOR\_PRODUTO

ou se input[índice] seja OPERADOR\_DIVISAO

índice++

retornar verdadeiro

retornar falso

função analisarOperadorRelacional() -> boolean

se input[índice] seja OPERADOR\_MENOR\_IGUAL

ou se input[índice] seja OPERADOR\_IGUAL

índice++

retornar verdadeiro

retornar falso

função analisarOperadorLogico() -> boolean

se input[índice] seja OPERADOR\_AND

ou se input[índice] seja OPERADOR\_OR

índice++

retornar verdadeiro

retornar falso

função analisarTipo() -> boolean

se input[índice] seja TIPO\_INTEIRO

ou se input[índice] seja TIPO\_REAL

ou se input[índice] seja TIPO\_CARACTER

ou se input[índice] seja TIPO\_CADEIA

ou se input[índice] seja TIPO\_LISTA\_INT

ou se input[índice] seja TIPO\_LISTA\_REAL

índice++

retornar verdadeiro

retornar falso