Projeto de Compiladores

Analisador 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”, “CARACTER”, “CADEIA”, “LISTA_INT” ou “LISTA_REAL”.

```
INTEIRO var1                    { número inteiro }
REAL var2                       { número real }
CARACTER 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 Algox

```
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
  ESCREVA 'fatorial =', fatorial
FIM
```

```
PROGRAMA leitura de lista
  INICIO
    { DECLARAÇÕES }
    INTEIRO n, i, x, k
    LISTA_INT A[100]
    { ALGORITMO }
    { Armazena os dados da lista }
    ESCREVA '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 }
    ESCREVA 'Número armazenados:'
    i := 1
    ENQUANTO i .M. n
      ESCREVA A[i], ' '
      i := i + 1
    FIM_ENQUANTO
  FIM
```

6. Gramática da linguagem Algox

Na gramática a seguir os não-terminais são representados entre < > 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.

$\mathbf{G_{ALGOX}}$ = ({ <algox>, <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_arimetico>, <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, VARIAVEL, VARIAVEL_LISTA, CADEIA, COMENTARIO }, P_{ALGOX}, <algox>)

$\mathbf{P_{ALGOX}}$ = {

<algox> → PROGRAMA <nome_do_programa> INICIO <corpo_do_programa> FIM

<nome_do_programa> → VARIAVEL <nome_do_programa_aux>

<nome_do_programa_aux> → VARIAVEL <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_aux> | <comando_entrada> <comando_aux> | <comando_saida> <comando_aux> | <comando_selecao> <comando_aux> | <comando_repeticao> <comando_aux>

<comando_aux> → <comando> <comando_aux> | ε

<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> → VARIAVEL | VARIAVEL_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 VARIAVEL VARIAVEL VARIAVEL INICIO TIPO_INTEIRO VARIAVEL VIRGULA VARIAVEL	PROGRAMA VARIAVEL INICIO TIPO_INTEIRO VARIAVEL VIRGULA VARIAVEL LEIA 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(<algox>) = { PROGRAMA }
Prim(<nome_do_programa>) = { VARIAVEL }
Prim(<nome_do_programa_aux>) = { VARIAVEL, ε }
Prim(<corpo_do_programa>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracoes>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracoes_aux>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Prim(<declaracao>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL }
Prim(<declaracao_aux>) = { VIRGULA, ε }
Prim(<algoritmo>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO }
Prim(<algoritmo_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, ε }
Prim(<comando>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO }
Prim(<comando_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, ε }
Prim(<comando_atribuicao>) = { VARIAVEL, VARIAVEL_LISTA }
Prim(<comando_entrada>) = { LEIA }
Prim(<comando_entrada_aux>) = { VIRGULA, ε }
Prim(<comando_saida>) = { ESCREVA }
Prim(<comando_saida_aux>) = { VARIAVEL, VARIAVEL_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, VARIAVEL, VARIAVEL_LISTA, ABRE }
Prim(<expressao_aritmetica_aux>) = { OPERADOR_ADICAO,

OPERADOR_SUBTRACAO, OPERADOR_PRODUTO, OPERADOR_DIVISAO, ε }
Prim(<expressao_relacional>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL, VARIAVEL, VARIAVEL_LISTA, ABRE }
Prim(<expressao_logica>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL, VARIAVEL, VARIAVEL_LISTA, ABRE, OPERADOR_NOT }
Prim(<expressao_logica_aux>) = { OPERADOR_AND, OPERADOR_OR, ε }
Prim(<constante>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL }
Prim(<variavel>) = { VARIAVEL, VARIAVEL_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(<algox>) = { $ }
Seg(<nome_do_programa>) = { INICIO }
Seg(<nome_do_programa_aux>) = { INICIO }
Seg(<corpo_do_programa>) = { FIM }
Seg(<declaracoes>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO }
Seg(<declaracoes_aux>) = { TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, ε }
Seg(<declaracao>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Seg(<declaracao_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, TIPO_INTEIRO, TIPO_REAL, TIPO_CARACTER, TIPO_CADEIA, TIPO_LISTA_INT, TIPO_LISTA_REAL, ε }
Seg(<algoritmo>) = { FIM }
Seg(<algoritmo_aux>) = { FIM }
Seg(<comando>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, FIM, ε }
Seg(<comando_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, FIM, ε }
Seg(<comando_atribuicao>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_entrada>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_entrada_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA,

ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_aux>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_variavel_cadeia>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_saida_cadeia_variavel>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_selecao>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<comando_repeticao>) = { VARIAVEL, VARIAVEL_LISTA, LEIA, ESCREVA, SE, ENQUANTO, FIM_SE, FIM_ENQUANTO, ε }
Seg(<expressao_aritmetica>) = { VARIAVEL, VARIAVEL_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>) = { VARIAVEL, VARIAVEL_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, VARIAVEL, VARIAVEL_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, VARIAVEL, VARIAVEL_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, VARIAVEL, VARIAVEL_LISTA, ABRE }
Seg(<operador_relacional>) = { CONSTANTE_INTEIRA, CONSTANTE_REAL, VARIAVEL, VARIAVEL_LISTA, ABRE }
Seg(<operador_logico>) => { CONSTANTE_INTEIRA, CONSTANTE_REAL, VARIAVEL, VARIAVEL_LISTA, ABRE, OPERADOR_NOT }
Seg(<tipo>) = { VARIAVEL, VARIAVEL_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 Analisador Sintático em Python

O código do analisador sintático está disponível no repositório do projeto no GitHub. O código foi montado usando um dicionário para converter os terminais e não-terminais da gramática em IDs numéricos para facilitar a implementação.

8.3.1 Tabela de Tradução Token - ID

O ID 0 representa o final da cadeia (representado por '$'), os IDs de 1 a 31 são os não-terminais ordenados de forma alfabética e os de 50 a 83 os terminais, também em ordem alfabética. A seguir tabela de relação entre token e ID:

TOKEN	ID
$	0
<algoritmo>	1
<algoritmo_aux>	2
<algox>	3
<comando>	4
<comando_aux>	5
<comando_atribuicao>	6
<comando_entrada>	7
<comando_entrada_aux>	8
<comando_repeticao>	9
<comando_saida>	10
<comando_saida_aux>	11
<comando_saida_cadeia_variavel>	12
<comando_saida_variavel_cadeia>	13
<comando_selecao>	14
<constante>	15
<corpo_do_programa>	16
<declaracao>	17
<declaracao_aux>	18
<declaracoes>	19

<declaracoes_aux>	20
<expressao_aritmetica>	21
<expressao_aritmetica_aux>	22
<expressao_logica>	23
<expressao_logica_aux>	24
<expressao_relacional>	25
<nome_do_programa>	26
<nome_do_programa_aux>	27
<operador_aritmetico>	28
<operador_logico>	29
<operador_relacional>	30
<tipo>	31
<variavel>	32
ABRE	50
CADEIA	51
CONSTANTE_INTEIRA	52
CONSTANTE_REAL	53
ENQUANTO	54
ENTAO	55
ESCREVA	56
FECHA	57
FIM	58
FIM_ENQUANTO	59
FIM_SE	60
INICIO	61
LEIA	62
OPERADOR_ADICAO	63
OPERADOR_AND	64

OPERADOR_ATRIBUICAO	65
OPERADOR_DIVISAO	66
OPERADOR_IGUAL	67
OPERADOR_MENOR_IGUAL	68
OPERADOR_NOT	69
OPERADOR_OR	70
OPERADOR_PRODUTO	71
OPERADOR_SUBTRACAO	72
PROGRAMA	73
SE	74
TIPO_CADEIA	75
TIPO_CARACTER	76
TIPO_INTEIRO	77
TIPO_LISTA_INT	78
TIPO_LISTA_REAL	79
TIPO_REAL	80
VARIAVEL	81
VARIAVEL_LISTA	82
VIRGULA	83

8.3.2 Resultados da Execução

Os testes foram feitos usando a lista de tokens gerados no analisador léxico nos dois exemplos mostrados na Seção 5. Em ambos os casos, ocorreu o reconhecimento sem precisar tratar erros.