# Trabalho 3

# -- Compiladores --

## Analisador Sintático


**Integrantes:**

| | |
|---|---|
| Caio Gomes | 7239072 |
| Diego Gomes | 7656467 |
| Fernando Cury Gorodscy | 7152354 |
| Roberty Manzini Bertolo | 7573399 |
| Vanessa Apolinário Lima | 7239256 |

# Sumário

# Introdução

Esse documento tem como objetivo apresentar a gramática implementada no Analisador Sintático JavaCC, os respectivos casos de testes analisados e um pouco da estrutura desenvolvida sobre o trabalho.

O nome da nossa gramática chama-se **Meuphoria.** Esse nome surgiu de uma brincadeira em sala de aula, na qual nós tinhamos que criar um nome, e queríamos que fosse nossa. Daí alguém sugeriu o "Meuphoria" - "Nossa euforia".

Caberá a cada uma das seções explicar um pouco sobre o desenvolvimento e ideia da implementação dada.

# Gramática

A gramática designada encontra-se abaixo com a nossa adição/característica única.

**TERMINAIS:**

ALPHA ::= ["a"-"z", "A"-"Z"]
DIGIT ::= ["0"-"9"]
USCORE ::= "_"
EOL ::= "\n"

**TOKENS:**

**- Palavras reservadas:**

CASE ::= "case"
DO ::= "do"
END ::= "end"
EXPORT ::= "export"
GLOBAL ::= "global"
INCLUDE ::= "include"
PUBLIC ::= "public"
SWITCH ::= "switch"
UNTIL ::= "until"
AS ::= "as"
CONSTANT ::= "constant"
ELSE ::= "else"
ENTRY ::= "entry"
FALLTHRU ::= "fallthru"
GOTO ::= "goto"
LABEL ::= "label"
RETRY ::= "retry"
THEN ::= "then"
WHILE ::= "while"
BREAK ::= "break"
CONTINUE ::= "continue"
ELSEDEF ::= "elsedef"
ENUM ::= "enum"
IF ::= "if"
LOOP ::= "loop"
OVERRIDE ::= "override"
RETURN ::= "return"
TO ::= "to"
WITH ::= "with"

BY ::= "by"
DEPRECATE ::= "deprecate"
ELSIF ::= "elsif"
EXIT ::= "exit"
FUNCTION ::= "function"
IFDEF ::= "ifdef"
NAMESPACE ::= "namespace"
PROCEDURE ::= "procedure"
ROUTINE ::= "routine"
TYPE ::= "type"
WITHOUT ::= "without"
PROFILE ::= "profile"
PROFILE_TIME ::= "profile_time"
TRACE ::= "trace"
BATCH ::= "batch"
TYPE_CHECK ::= "type_check"
INDIRECT_INCLUDES ::= "indirect_includes"
INLINE ::= "inline"
WARNING ::= "warning"
SAVE ::= "save"
RESTORE ::= "restore"
STRICT ::= "strict"
ELSEIFDEF ::= "elsifdef"
FOR ::= "for"

**- Separadores:**

LPAREN ::= "("
RPAREN ::= ")"
LBRACE ::= "{"
RBRACE ::= "}"
LBRACKET ::= "["
RBRACKET ::= "]"
SEMICOLON ::= ";"
COMMA ::= ","
DOT ::= "."
#SINGLE_QUOTE ::= "\'"
SLICE ::= ".."
#QUOTE ::= "\""
#TRIPLE_QUOTE ::= "\"\"\""
COLON ::= ":"

**- Operadores:**

NOT_OP ::= "not"
PLUS ::= "+"
MINUS ::= "-"
STAR ::= "*"
SLASH ::= "/"

CONCAT ::= "&"
GT ::= ">"
LT ::= "<"
LE ::= "<="
GE ::= ">="
EQUAL ::= "="
NE ::= "!="
SC_OR ::= "or"
SC_AND ::= "and"
XOR_OP ::= "xor"
QMARK ::= "?"
END_SYMBOL ::= "$"

**MANIPULAÇÃO VARIÁVEIS:**

IDENTIFIER ::= ( ALPHA | USCORE ) { ALPHA | DIGIT | USCORE }
ATOM ::= INTEGER | REAL

INTEGER ::= [ PLUS | MINUS ] DIGIT { DIGIT | USCORE }
REAL ::= FLOAT | EXP

FLOAT ::= INTEGER DOT DIGIT { DIGIT | USCORE }
EXP ::= ( INTEGER | FLOAT ) ("E" | "e") INTEGER

SEQUENCE ::= LBRACE OBJECT { COMMA OBJECT } [ COMMA END_SYMBOL ] RBRACE

OBJECT ::= ATOM | SEQUENCE | BIN_NUMBER | OCT_NUMBER | DEC_NUMBER | HEX_NUMBER

BIN_NUMBER ::= "0b" BIN

BIN ::= ("0" | "1") { "0" | "1" }

OCT_NUMBER ::= "0t" OCT

OCT ::= ["0" - "7"] { ["0" - "7"] }

DEC_NUMBER ::= "0d" INTEGER

HEX_NUMBER ::= "0x" HEX

HEX ::= ["0" - "9", "A" - "F"] { ["0" - "9", "A" - "F"] }

PARAMLIST ::= PARAMETER {COMMA PARAMETER }

PARAMETER ::= DATATYPE IDENTIFIER

IDENTLIST ::= IDENT { COMMA IDENT }

IDENT ::= IDENTIFIER [ EQUAL EXPRESSION ]

VARIABLE ::= IDENTIFIER [ SLICESTMT | SUBSCRIPTING ]

SLICESTMT ::= LBRACKET EXPRESSION SLICE ( EXPRESSION | END_SYMBOL) RBRACKET

SUBSCRIPTING ::= INDEX { INDEX }

INDEX ::= LBRACKET EXPRESSION RBRACKET

**MANIPULAÇÃO DE STRINGS:**

CHAR = ["a"-"z", "A"-"Z"]

STRINGLIT ::= SIMPLESTRINGLIT | RAWSTRINGLIT

SIMPLESTRINGLIT ::= QUOTE { CHAR | ESCCHAR } QUOTE

ESCCHAR ::= ESCLEAD ( "n" | "r" | "t" | "\\" | "\"" | "\'" | "0" | "e" | "E" | BINARY | HEX2 | HEX4 | HEX8 )

BINARY ::= "\\b" BIN
HEX2 ::= "\\x" ["0" - "7"] ["0" - "7"]
HEX4 ::= "\\u" ["0" - "7"] ["0" - "7"]["0" - "7"] ["0" - "7"]
HEX8 ::= "\\U" ["0" - "7"] ["0" - "7"]["0" - "7"] ["0" - "7"]["0" - "7"] ["0" - "7"]["0" - "7"] ["0" - "7"]

RAWSTRINGLIT ::= DQRAWSTRING | BQRAWSTRING | VANSTRING
DQRAWSTRING ::= "`" CHAR {CHAR} "`"
VANSTRING ::= "’" CHAR {CHAR}  "’"
BQRAWSTRING ::= "\"\"\"" CHAR {CHAR} "\"\"\""

HEX = ([“0” - “9”][“A” - “F”])*
DEC = ([“0” - “9”])*
OCT = ([“0” - “7”])*

LABELSTMT ::= LABEL STRINGLIT

**DATA TYPE:**

SCOPEMODIFEIR ::= GLOBAL | PUBLIC | EXPORT | OVERRIDE
DATATYPE ::= "atom" | "integer" | "sequence" | "object" | IDENTIFIER

**EXPRESSÕES:**

EXPRESSION ::= SEQUENCE | E0
E0 ::=  E1 { (SC_AND | SC_OR | SC_XOR) E1}
E1 ::= E2 { RELATIONOP E2 }
RELATIONOP ::= "<" | ">" | "<=" | ">=" | "=" | "!="

E2 ::= E3 { CONCAT E3 }
E3 ::= E4 { (PLUS | MINUS) E4 }
E4 ::= E5 { (STAR | SLASH) E5 }
E5 ::= [PLUS | MINUS | NOT_OP] E6
E6 ::= ATOM | STRINGLIT | SEQUENCE |VARIABLE | "0" | "1" | (LPAREN EXPRESSION RPAREN) | IDENTIFIER | CALL

CALL ::= IDENTIFIER "(" [ ARGLIST ] ")"
ARGLIST ::= EXPRESSION { "," EXPRESSION }

**STATEMENT:**

STMBLK ::= STATEMENT {STATEMENT}
STATEMENT ::= FLOW | LOOP | BRANCH | ASSIGNMENTSTMT | RETURN | VARDECLARE | CONSTDECLARE | ENUMDECLARE | PROCDECLARE | FUNCDECLARE | TYPEDECLARE | CALL

**- Statements Básicos:**

WITHENTRY ::= WITH ENTRY
ENTRYSTMT ::= ENTRY [STMBLK]

**- Controle de Fluxo:**

FLOW ::= BREAKSTMT | CONTINUESTMT | RETRYSTMT | EXITSTMT | FALLTHRUSTMT

BREAKSTMT ::= BREAK [STRINGLIT]
CONTINUESTMT ::= CONTINUE [STRINGLIT]
RETRYSTMT ::= RETRY [STRINGLIT]
EXITSTMT ::= EXIT [STRINGLIT]
FALLTHRUSTMT ::= FALLTHRU [STRINGLIT]

**- Loop:**

LOOP ::= FORSTMT | WHILESTMT | LOOPSTMT | GOTOSTMT

FORSTMT ::= FOR FORIDX [LABEL] DO [STMBLK] END FOR

FORIDX ::= IDENTIFIER EQUAL EXPRESSION TO EXPRESSION [BY EXPRESSION]

WHILESTMT ::= WHILE EXPRESSION [WITHENTRY][LABEL] DO STMBLK [ENTRYSTMT] END WHILE

LOOPSTMT ::= LOOP [WITHENTRY][LABEL] DO STMBLK [ENTRYSTMT] UNTIL EXPRESSION END LOOP

GOTOSTMT ::= GOTO LABEL

- Branching:

BRANCH ::= IFSTMT | SWITCHSTMT | IFDEFSTMT

IFSTMT ::= IFTEST {ELSIFSTMT} [ELSESTMT] ENDIF

IFTEST ::= IF EXPRESSION [LABEL] THEN [STMBLK]
ELSIFSTMT ::= ELSIF EXPRESSION THEN [STMBLK]
ELSESTMT ::= ELSE [STMBLK]
ENDIF ::= END IF

SWITCHSTMT ::= SWITCHTEST CASESTMT {CASESTMT} [CASEELSE] ENDSWITCH

SWITCHTEST ::= SWITCH EXPRESSION [WITHFALL] [LABEL] DO

WITHFALL ::= (WITH | WITHOUT) FALLTHRU

CASESTMT ::= CASE CASELIST THEN [ STMBLK ]
CASELIST ::= EXPRESSION {"," EXPRESSION }
CASEELSE ::= ELSE [STMBLK]

ENDSWITCH ::= END SWITCH

IFDEFSTMT ::= IFDEFTEST [ELSDEFIFSTMT {CHAR}][ELSEDEFSTMT]ENDDEFIF

IFDEFTEST ::= IFDEF DEFEXPR THEN [STMBLK]
ELSDEFIFSTMT ::= ELSEIFDEF DEFEXPR THEN [STMBLK]
ELSEDEFSTMT ::= ELSEDEF [STMBLK]
ENDDEFIF ::= END IFDEF
DEFEXPR ::= DEFTERM [ DEFOP DEFTERM ]
DEFTERM ::= [ NOT ] IDENTIFIER

DEFOP ::= SC_AND | SC_OR

**- Atribuição:**

ASSIGNMENTSTMT ::= ASSIGNMONO | ASSIGNMULTI | ASSIGNWITHOP

ASSIGNMONO ::= IDENTIFIER EQUAL EXPRESSION
ASSIGNMULTI ::= IDENTIFIER {COMMA IDENTIFIER} EQUAL EXPRESSION {COMMA EXPRESSION}
ASSIGNWITHOP ::= IDENTIFIER ( PLUS | MINUS | SLASH | STAR | CONCAT ) EQUAL EXPRESSION

**- Return:**

RETURNSTMT ::= RETURN EXPRESSION { COMMA EXPRESSION } ***--Característica Meuphoria***

**- Declaração de Variáveis:**

VARDECLARE ::= [ SCOPEMODIFIER ] DATATYPE IDENTLIST

**- Declaração de Constantes:**

CONSTDECLARE **::=** [ SCOPEMODIFIER ] CONSTANT IDENTLIST

**- Declaração de Enumeração:**

ENUMDECLARE **::=** [SCOPEMODIFIER] (ENUMVAL|ENUMTYPE)

ENUMVAL **::=** ENUM IDENTLIST

ENUMTYPE **::=** ENUM TYPE IDENTLIST END TYPE

**- Declaração de Processos:**

PROCDECLARE **::=** [ SCOPEMODIFIER ] PROCEDURE IDENTIFIER LPAREN [ PARAMLIST ] RPAREN [STMBLK] END PROCEDURE

**- Declaração de Função:**

FUNCDECLARE **::=** [ SCOPEMODIFIER ] FUNCTION IDENTIFIER LPAREN [PARAMLIST] RPAREN [STMBLK] END FUNCTION

**- Declaração de Tipo:**

TYPEDECLARE **::=** [ SCOPEMODIFIER ] TYPE IDENTIFIER LPAREN [PARAMLIST] RPAREN [STMBLK] END TYPE

# Casos de Testes

## Happy Tests – Testes certos (eram para dar sucesso)

**Teste: Comentário**

**Entrada:**

```
-- Testando comentario.
-- Testando comentário com acentos e alguns caracteres especiais áéíóúç~`!@#$%^&*()-+=.
puts(1, "Testando a Meuphoria!\n") -- testando comentário após código
/* testando um comentário de bloco
pra ver se o compilador pode
ignorar tudo isso sem problemas*/
```

**Saída:**

```
SUCESSO!
```

**Teste: Declarar Variáveis**

**Entrada:**

```
sequence nome
integer tempo
integer miles
```

**Saída:**

```
SUCESSO!
```

**Teste: Atribuir Valores**

**Entrada:**

```
integer tempo, distancia
sequence nome

tempo   = 100
tempo   = 101
distancia = 1000.0              -- Setar um ponto flutuante
nome    = "Vanessa"             -- Setar uma String
constant MAX = 100              -- Setar uma constante
enum ONE, TWO, THREE, FOUR      -- Enumerar
```

**Saída:**

SUCESSO!

**Teste: Imprimir variáveis**

**Entrada:**

```
integer tempo, distancia
sequence nome

tempo   = 100                   -- Setar um integer
tempo   = 101                   -- Tentar substituir valor
distancia = 1000.0              -- Setar um ponto flutuante
nome    = "Vanessa"             -- Setar uma String
constant MAX = 100              -- Setar uma constante
enum ONE, TWO, THREE, FOUR          -- Enumerar

printf(1, "Valor de tempo %d\n", tempo)
printf(1, "Valor de miles %f\n", distancia )
printf(1, "Valor de nome %s\n", {nome} )
printf(1, "Valor de ONE %d\n", ONE )
printf(1, "Valor de TWO %d\n", TWO )
printf(1, "Valor de THREE %d\n", THREE )
printf(1, "Valor de FOUR %d\n", FOUR )
```

**Saída:**

SUCESSO!

**Teste: Chaves Sequencia**

**Entrada:**

```
enum X, Y
sequence point = { 0,0 }
point[X] = 3
point[Y] = 4

printf(1, "Valor da Sequencia %d %d\n", point)
```

**Saída:**

SUCESSO!

**Teste: Operadores Aritméticos**

**Entrada:**

```
integer a = 10
integer b = 20

printf(1, "a + b = %d\n" , (a + b) )
printf(1, "a - b = %d\n" , (a - b) )
printf(1, "a * b = %d\n" , (a * b) )
printf(1, "b / a = %d\n" , (b / a) )
printf(1, "+a   = %d\n" ,  (+a) )
printf(1, "-a   = %d\n" ,  (-a) )
printf(1, "a = b = %d\n", (a = b) )
printf(1, "a != b = %d\n", (a != b) )
printf(1, "a > b = %d\n", (a > b) )
printf(1, "a < b = %d\n", (a < b) )
printf(1, "b >= a = %d\n", (b >= a) )
printf(1, "b <= a = %d\n", (b <= a) )
```

**Saída:**

```
SUCESSO!
```

**Teste: Operadores Lógicos**

**Entrada:**

```
integer a = 1
integer b = 0
integer c = 1

printf(1, "a and b = %d\n", (a and b) )
printf(1, "a or b = %d\n", (a or b) )
printf(1, "a xor b = %d\n", (a xor b) )
printf(1, "a xor c = %d\n", (a xor c) )
printf(1, "not(a) = %d\n", not(a) )
printf(1, "not(b) = %d\n", not(b) )
```

**Saída:**

```
SUCESSO!
```

**Teste: Operadores Atribuicao**

**Entrada:**

```
integer a = 10
integer b = 20
integer c = 0
```

```
c = a + b
printf(1, "c = a + b = %d\n", c )

c += a
printf(1, "c += a = %d\n", c )

c -= a
printf(1, "c -= a = %d\n", c )

c *= a
printf(1, "c *= a = %d\n", c )

a = 10
c = 30
c /= a
printf(1, "c /= a = %d\n", c )
```

**Saída:**

```
SUCESSO!
```

**Teste: If Statement**

**Entrada:**

```
integer a = 10
integer b = 20

if (a + b) < 40 then
  printf(1, "%s\n", {"Entrei nesse statement!"})
end if

if (a + b) > 40 then
  printf(1, "%s\n", {"Nao entrei aqui"})
else
  printf(1, "%s\n", {"Aqui sim!"})
end if
```

**Saída:**

```
SUCESSO!
```

**Teste: Switch Statement**

**Entrada:**

```
atom nota = 'C'
```

```
switch nota do
  case 'A' then
    puts(1, "Excelente!\n" )
  case 'B', 'C' then
    puts(1, "Muito bom!\n" )
  case 'D' then
    puts(1, "Aprovado!\n" )
  case 'F' then
    puts(1, "Oh, nao! Fui reprovado!\n" )
  case else
    puts(1, "Nota invalida!\n" )
end switch
```

**Saída:**

```
SUCESSO!
```

**Teste: While Statement**

**Entrada:**

```
integer a = 10
while a < 20 do
  printf(1, "Valor de a : %d\n", a)
  a = a + 1
end while
```

**Saída:**

```
SUCESSO!
```

**Teste: For Statement**

**Entrada:**

```
for a = 1 to 6 do
  printf(1, "valor de a %d\n", a)
end for
```

**Saída:**

```
SUCESSO!
```

**Teste: Flow Statement**

**Entrada:**

```
integer b


for a = 1 to 16 do
  printf(1, "valor de a: %d\n", a)
  if a = 10 then
    b = a
    exit
  end if
end for

printf(1, "valor de b vindo de a: %d\n", b)
```

**Saída:**

```
SUCESSO!
```

**Teste: Flow Break**

**Entrada:**

```
integer a, b
sequence s = {'E','u', 'p'}


if s[1] = 'E' then
  a = 3
  if s[2] = 'u' then
    b = 1
    if s[3] = 'p' then
      break 0
    end if
    a = 2
  else
    b = 4
  end if
else
  a = 0
  b = 0
end if


printf(1, "valor de a %d\n", a)
printf(1, "valor de b %d\n", b)
```

**Saída:**

```
SUCESSO!
```

**Teste: Flow Continue**

**Entrada:**

```
for a = 3 to 6 do
  printf(1, "valor de a %d\n", a)
  if a = 4 then
    puts(1,"(2)\n")
    continue
  end if
  printf(1, "valor de a %d\n", a*a)
end for
```

**Saída:**

```
SUCESSO!
```

**Teste: Procedure**

**Entrada:**

```
procedure DigaOi(sequence nome,atom  idade)
  printf(1, "%s tem %d anos.", {nome, idade})
end procedure
-- chama a procedure definida em cima.
DigaOi("zara", 8)
```

**Saída:**

```
SUCESSO!
```

**Teste: Function**

**Entrada:**

```
function DigaOi()
  puts(1, "Oi Professora")
  return 1
end function


-- Chama a função acima
DigaOi()
```

**Saída:**

SUCESSO!

**Teste: Our Function**

**Entrada:**

```
sequence nome
integer idade


nome, idade = "Vanessa", 10
```

**Saída:**

SUCESSO!

**Teste: Escaped**

**Entrada:**

```
printf(1, "Valor de 1 \n")
printf(1, "Valor de 2 \r")
printf(1, "Valor de 3 \t")
printf(1, "Valor de 4 \\")
printf(1, "Valor de 5 \"")
printf(1, "Valor de 6 \'")
printf(1, "Valor de 7 \0")
printf(1, "Valor de 8 \e")
printf(1, "Valor de 9 \E")
printf(1, "Valor de 10 \x5F")
printf(1, "Valor de 11 \u2A7C")
printf(1, "Valor de 12 \U8123FEDC")
```

**Saída:**

SUCESSO!

# Sad Tests – Testes com erros (eram para dar erros)

**Teste: Variável Acentuada**

**Entrada:**

```
integer distância
distância = 1000
```

```
printf(1, "Valor de distância %f\n", distância)
```

**Saída:**

```
Lexical error at line 1, column 13.  Encountered: "\ufffd" (65533), after : ""
Existem erros no codigo! Por favor, verifica-los.
```

**Teste: Variavel com caracter especial**

**Entrada:**

```
integer dist@ncia
dist@ncia = 1000
printf(1, "Valor de dist@ncia %f\n", dist@ncia)
```

**Saída:**

```
Encountered " <CHAR> "@ "" at line 1, column 13.

Was expecting one of:
   <EOF>
   "\n" ...
   "export" ...
   "global" ...
  "public" ...
   "switch" ...
   "constant" ...
   "fallthru" ...
   "goto" ...
   "retry" ...
   "while" ...
   "break" ...
   "continue" ...
   "enum" ...
   "if" ...
   "loop" ...
   "override" ...
   "return" ...
   "exit" ...
   "function" ...
   "ifdef" ...
   "procedure" ...
   "type" ...
   "for" ...
   "," ...
   "=" ...
   "atom" ...
   "integer" ...
   "sequence" ...
```

```
  "object" ...
  <IDENTIFIER> ...

Ocorreu uma excecao!
```

**Teste: String Aberta**

**Entrada:**

```
integer tempo, distancia
sequence nome


tempo   = 100                   -- Setar um integer
tempo   = 101                   -- Tentar substituir valor
distancia = 1000.0              -- Setar um ponto flutuante
nome    = "Vanessa              -- Setar uma String
constant MAX = 100              -- Setar uma constante
enum ONE, TWO, THREE, FOUR          -- Enumerar
```

**Saída:**

```
Lexical error at line 7, column 24.  Encountered: "\t" (9), after : "\"Vanessa    "
Existem erros no codigo! Por favor, verifica-los.
```

**Teste: Variavel com Caracter Especial**

**Entrada:**

```
        integer t&mpo
        t&mpo = 1000
        printf(1, "Valor de t&mpo %f\n", t&mpo)
```

**Saída:**

```
Encountered " "&" "& "" at line 1, column 10.
Was expecting one of:
  <EOF>
  "\n" ...
  "export" ...
  "global" ...
  "public" ...
  "switch" ...
  "constant" ...
  "fallthru" ...
  "goto" ...
  "retry" ...
  "while" ...
  "break" ...
```

```
  "continue" ...
  "enum" ...
  "if" ...
  "loop" ...
  "override" ...
  "return" ...
  "exit" ...
  "function" ...
  "ifdef" ...
  "procedure" ...
  "type" ...
  "for" ...
  "," ...
  "=" ...
  "atom" ...
  "integer" ...
  "sequence" ...
  "object" ...
 <IDENTIFIER> ...

Ocorreu uma excecao!
```

**Teste: Variavel com caracter especial**

**Entrada:**

```
integer teste%teste
teste%teste = 1000
printf(1, "Valor de teste %f\n", teste%teste)
```

**Saída:**

```
Encountered " <CHAR> "% "" at line 1, column 14.
Was expecting one of:
  <EOF>
  "\n" ...
  "export" ...
  "global" ...
  "public" ...
  "switch" ...
  "constant" ...
  "fallthru" ...
  "goto" ...
  "retry" ...
  "while" ...
  "break" ...
  "continue" ...
```

"enum" ...
"if" ...
"loop" ...
"override" ...
"return" ...
"exit" ...
"function" ...
"ifdef" ...
"procedure" ...
"type" ...
"for" ...
"," ...
"=" ...
"atom" ...
"integer" ...
"sequence" ...
"object" ...
<IDENTIFIER> ...

Ocorreu uma excecao!

**Teste: Numero na Variavel**

**Entrada:**

```
integer 5distância
distância = 1000
printf(1, "Valor de distância %f\n", 5distância)
```

**Saída:**

```
Encountered " "integer" "integer "" at line 1, column 1.
Was expecting one of:
   "export" ...
   "global" ...
   "public" ...
   "constant" ...
   "enum" ...
   "override" ...
   "function" ...
   "procedure" ...
   "global" ...
   "public" ...
   "export" ...
  "override" ...
```

Ocorreu uma excecao!

**Teste: Comentário aberto**

**Entrada:**

```
-- Testando comentario.
-- Testando comentário com acentos e alguns caracteres especiais áéíóúç~`!@#$%^&*()-+=.

/* testando um comentário de bloco
pra ver se o compilador pode
ignorar tudo isso sem problemas


puts(1, "Testando a Meuphoria!\n") -- testando comentário após código
```

**Saída:**

```
Lexical error at line 8, column 70.  Encountered: <EOF> after : ""
Existem erros no codigo! Por favor, verifica-los.
```

**Teste: Variavel 30 Caracteres**

**Entrada:**

```
integer EssaVariavelTemMaisDeTrintaCaracteres


EssaVariavelTemMaisDeTrintaCaracteres = 1000


printf(1, "Valor de variavel %f\n", EssaVariavelTemMaisDeTrintaCaracteres)
```

**Saída:**

```
null
Existem erros no codigo! Por favor, verifica-los.
```

**Teste: Variavel 30 Caracteres**

**Entrada:**

```
sequence EssaVariavelTemMaisDeTrintaCaracteres


EssaVariavelTemMaisDeTrintaCaracteres = "teste"
```

```
printf(1, "Valor de variavel %s\n", {EssaVariavelTemMaisDeTrintaCaracteres})
```

**Saída:**

```
null
Existem erros no codigo! Por favor, verifica-los.
```

**Teste: If aberto**

**Entrada:**

```
integer a = 10
integer b = 20

if (a + b) < 40 then
  printf(1, "%s\n", {"Entrei nesse statement!"})
end if

if (a + b > 40 then
  printf(1, "%s\n", {"Não entrei aqui"})
else
  printf(1, "%s\n", {"Aqui sim!"})
end if
```

**Saída:**

```
Encountered " "then" "then "" at line 8, column 16.
Was expecting one of:
   ")" ...
   "+" ...
   "-" ...
   "*" ...
   "/" ...
   "&" ...
   ">" ...
   "<" ...
   "<=" ...
   ">=" ...
   "=" ...
   "!=" ...
   "or" ...
   "and" ...
   "xor" ...

Ocorreu uma excecao!
```

**Teste: Sequence**

**Entrada:**

```
sequence a = 0b012
```

**Saída:**

```
Encountered " <BIN_NUMBER> "0b01 "" at line 1, column 14.
Was expecting one of:
    "(" ...
    "{" ...
    "not" ...
    "+" ...
    "-" ...
    <IDENTIFIER> ...
    <INTEGER> ...
    <FLOAT> ...
    <EXP> ...
    <STRINGLIT> ...


Ocorreu uma excecao!
```

**Teste: While**

**Entrada:**

```
integer a = 10

while a < 20
  printf(1, "Valor de a : %d\n", a)
  a = a + 1
end while
```

**Saída:**

```
Encountered " "\n" "\n "" at line 3, column 13.
Was expecting one of:
    "do" ...
    "+" ...
    "-" ...
    "*" ...
    "/" ...
    "&" ...
    ">" ...
    "<" ...
    "<=" ...
    ">=" ...
    "=" ...
    "!=" ...
```

```
   "or" ...
   "and" ...
  "xor" ...
   <WITHENTRY> ...
   <LABELSTMT> ...
```

```
Ocorreu uma excecao!
```

**Teste: Chaves Sequencia**

**Entrada:**

```
enum X, Y
sequence point = { 0,0
point[X] = 3
point[Y] = 4


printf(1, "Valor da Sequencia %d %d\n", point)
```

**Saída:**

```
Encountered " "\n" "\n "" at line 2, column 24.
Was expecting one of:
   "}" ...
   "," ...
   "," ...
```

```
Ocorreu uma excecao!
```

**Teste: For Aberto**

**Entrada:**

```
for a = 1 to c
  printf(1, "valor de a %d\n", a)
end for
```

**Saída:**

```
Encountered " "\n" "\n "" at line 1, column 15.
Was expecting one of:
   "do" ...
   "by" ...
   "(" ...
   "[" ...
   "+" ...
   "-" ...
   "*" ...
```

```
  "/" ...
  "&" ...
  ">" ...
  "<" ...
  "<=" ...
  ">=" ...
  "=" ...
  "!=" ...
  "or" ...
  "and" ...
  "xor" ...
  <LABELSTMT> ...

Ocorreu uma excecao!
```

**Teste: For Sem Fechar**

**Entrada:**

```
for a = 1 to 5 do
  printf(1, "valor de a %d\n", a)
```

**Saída:**

```
Encountered "<EOF>" at line 2, column 34.
Was expecting one of:
  "\n" ...
  "end" ...
  "export" ...
  "global" ...
  "public" ...
  "switch" ...
  "fallthru" ...
 "goto" ...
  "retry" ...
  "while" ...
  "break" ...
  "continue" ...
  "if" ...
  "loop" ...
  "override" ...
  "return" ...
  "exit" ...
  "ifdef" ...
  "type" ...
  "for" ...
  "atom" ...
  "integer" ...
```

"sequence" ...
"object" ...
<IDENTIFIER> ...

Ocorreu uma excecao!

# Análise rápida dos resultados e Conclusão

Em suma, atingimos os resultados esperados como pode-se ver pelos casos de testes que conseguem efetivar todos os erros e acertos léxicos e sintáticos que pudemos imaginar, entretanto, o trabalho demandou muito mais trabalho do que o previsto ou suposto.