**UNIVERSIDADE FEDERAL DE ALAGOAS**
**INSTITUTO DE COMPUTAÇÃO**
**CIÊNCIA DA COMPUTAÇÃO**

PEDRO JAVIER PANECA CORDOVA

# Compiladores

Especificação dos Tokens

MACEIÓ/AL
2021.1

## Linguagem utilizada para a criação do Analisador Léxico e Sintático:

Python

## Enumeração com as Categorias dos Tokens:

```
from enum import Enum, auto

class TokenCategory(Enum):
 # Identifier
 identifier = auto()

 # Primitives Types
 typeInt = auto()
 typeFloat = auto()
 typeBool = auto()
 typeChar = auto()
 typeString = auto()
 typeVoid = auto()

 # Accepted Values by the Primitives Types
 boolVal = auto()
 intVal = auto()
 floatVal = auto()
 charVal = auto()
 stringVal = auto()

 # Conditional Structure Commands
 cmdIf = auto()
 cmdElsif = auto()
 cmdElse = auto()

 # Repetition Structure Commands
 cmdFor = auto()
```

```
cmdWhile = auto()

# Logics Operators
opTrue = auto()
opFalse = auto()
opNot = auto()
opOr = auto()
opAnd = auto()

# Relationals Operators
opEqual = auto()
opNotEqual = auto()
opGtrThan = auto()
opLessThan = auto()
opGtrEqual = auto()
opLessEq = auto()

# Algebraic Operators
opAttr = auto()
opAdd = auto()
opSub = auto()
opMult = auto()
opDiv = auto()
opMod = auto()
opUnaryNeg = auto()
opUnaryPos = auto()

# Concatenation Operator
opConcat = auto()

# Program Start Execution Point
main = auto()

# Terminal
```

```
semicolon = auto()

# Separators
commaSep = auto()

# Functions
fnDecl = auto()
fnRtn = auto()

# Input/Output
fnRead = auto()
fnWrite = auto()

# One-Dimensional Arrays
arrayBegin = auto()
arrayEnd = auto()

# Params Delimiters
paramBegin = auto()
paramEnd = auto()

# Scope Delimiters
scopeBegin = auto()
scopeEnd = auto()

# Tokens not defined
tokenNotDefined = auto()

# End Of File
EOF = auto()
```

**Expressões regulares:**

varName: [‘letters’]([‘letters’’digits’_]*)

**Expressões regulares auxiliares:**

letters: [a-zA-z]

digits: [0-9]

specialCharacters: ‘ ‘ | ‘\`‘ | ‘\~’ | ‘\!’ | ‘\@’ | ‘\#’ | ‘\$’ | ‘\%’ | ‘\^’ | ‘\&’ | ‘*’ | ‘(’ | ‘)’ | ‘\-’ | ‘_’ | ‘\+’ | ‘\=’ | ‘\’ | ‘\{’ | ‘\}’ | ‘\[’ | ‘\]’ | ‘\|’ | ‘\\’ | ‘\;’ | ‘\:’ | ‘\”’ | ‘\’’ | ‘\?’ | ‘\/’ | ‘\´’

**Tipos primitivos:**

typeInt = ‘int’
typeFloat = ‘float’
typeBool = ‘bool’
typeChar = ‘char’
typeString = ‘string’
typeVoid = ‘void’

**Valores aceitos pelos tipos primitivos:**

boolVal: (‘True’ | ‘False’)

intVal: [+-]?(‘digits’)+

floatVal: [+-]?((digits)+[.])?(‘digits’+)

charVal: (‘\’’)(^[‘letters’’numbers’’specialCharacters’]$)(‘\’’)

stringVal: (‘\”’)([‘letters’ | ‘digits’ | ‘specialCharacters’])*(‘\”’)

**Comandos para estrutura condicional:**

cmdIf: ‘if’
cmdElsif: ‘elsif’
cmdElse: ‘else’

## Comandos para estrutura de repetição:

cmdFor: ‘for’
cmdWhile: ‘while’

## Operadores lógicos:

opTrue: ‘True’
opFalse: ‘False’
opNot: ‘!’
opOr: ‘||’
opAnd: ‘&&’

## Operadores relacionais:

opEqual: ‘==’
opNotEq: ‘!=’
opGtrThan: ‘>’
opLessThan: ‘<’
opGtrEqual: ‘>=’
opLessEq: ‘<=’

## Operadores matemáticos:

opAtt: ‘=’
opAdd: ‘+’
opSub: ‘-’
opMult: ‘*’
opDiv: ‘/’
opMod: ‘%’
opUnaryNeg: ‘-’
opUnaryPos: ‘+’

## Operador de concatenação:

opConcat: ‘.’

**Ponto inicial de execução do programa:**

main: ‘main’

**Terminais:**

semicolon: ‘;’

**Separadores:**

commaSep: ‘,’

**Funções:**

fnDecl: ‘fn’
fnRtn: ‘return’

**Leitura e escrita:**

fnRead: ‘gets’
fnWrite: ‘puts’

**Arranjos unidimensionais:**

arrayBegin: ‘\[‘
arrayEd: ‘\]’

**Delimitadores de parâmetros:**

paramBegin: ‘\(‘
paramEnd: ‘\)’

**Delimitadores de escopo:**

scopeBegin: ‘\{‘
scopeEnd: ‘\}’

**Tokens não identificados pela linguagem:**

tokenNotDefined

| Valor numérico da Categoria do Token | Nome simbólico do Token | Expressão Regular do Lexema |
|---|---|---|
| 1 | varName | [‘letters’]([‘letters’’digits’_]*) |
| 2 | typeInt | ‘int’ |
| 3 | typeFloat | ‘float’ |
| 4 | typeBool | ‘bool’ |
| 5 | typeChar | ‘char’ |
| 6 | typeString | ‘string’ |
| 7 | typeVoid | ‘void’ |
| 8 | boolVal | (‘True’ \| ‘False’) |
| 9 | intVal | [+-]?(‘digits’)+ |
| 10 | floatVal | [+-]?((digits)+[.])?(‘digits’+) |
| 11 | charVal | (‘\’’)(^[‘letters’’numbers’’specialCharacters’]$)(‘\’’) |
| 12 | stringVal | (‘\”’)([‘letters’ \| ‘digits’ \| ‘specialCharacters’])*(‘\”’) |
| 13 | cmdIf | ‘if’ |
| 14 | cmdElsif | ‘elsif’ |
| 15 | cmdElse | ‘else’ |
| 16 | cmdFor | ‘for’ |
| 17 | cmdWhile | ‘while’ |
| 18 | opTrue | ‘True’ |
| 19 | opFalse | ‘False’ |
| 20 | opNot | ‘!’ |
| 21 | opOr | ‘\|\|’ |

| | | |
|---|---|---|
| 22 | opAnd | ‘&&’ |
| 23 | opEqual | ‘==’ |
| 24 | opNotEqual | ‘!=’ |
| 25 | opGtrThan | ‘>’ |
| 26 | opLessThan | ‘<’ |
| 27 | opGtrEqual | ‘>=’ |
| 28 | opLessEq | ‘<=’ |
| 29 | opAttr | ‘=’ |
| 30 | opAdd | ‘+’ |
| 31 | opSub | ‘-’ |
| 32 | opMult | ‘*’ |
| 33 | opDiv | ‘/’ |
| 34 | opMod | ‘%’ |
| 35 | opUnaryNeg | ‘-’ |
| 36 | opUnaryPos | ‘+’ |
| 37 | opConcat | ‘.’ |
| 38 | main | ‘main’ |
| 39 | semicolon | ‘;’ |
| 40 | commaSep | ‘,’ |
| 41 | fnDecl | ‘fn’ |
| 42 | fnRtn | ‘return’ |
| 43 | fnRead | ‘gets’ |
| 44 | fnWrite | ‘puts’ |
| 45 | arrayBegin | ‘\[‘ |

|  |  |  |
|---|---|---|
| 46 | arrayEnd | ‘\]’ |
| 47 | paramBegin | ‘\(‘ |
| 48 | paramEnd | ‘\)’ |
| 49 | scopeBegin | ‘\{‘ |
| 50 | scopeEnd | ‘\}’ |
| 51 | tokenNotDefined |  |