

**Universidade Federal de Alagoas**  
**Instituto de computação**

**Compiladores**

Especificação dos tokens

**Alunos:** Davi José de Melo Silva, Gustavo Augusto Calazans Lopes

**Professor:** Alcino Dall'Igna Jr.

2020.2

## **Linguagem utilizada para criar o analisador léxico:**

A linguagem escolhida para a implementação foi Python.

## **Enumeração com as categorias dos Tokens na linguagem utilizada:**

```
from enum import Enum, auto
```

```
class TokensCategories(Enum):
```

```
    identifier = auto()
```

```
    typeInt = auto()
```

```
    typeFloat = auto()
```

```
    typeBool = auto()
```

```
    typeChar = auto()
```

```
    typeString = auto()
```

```
    typeVoid = auto()
```

```
    intVal = auto()
```

```
    floatVal = auto()
```

```
    boolVal = auto()
```

```
    charVal = auto()
```

```
    stringVal = auto()
```

```
    main = auto()
```

```
    cmdFor = auto()
```

```
    cmdWhile = auto()
```

```
    cmdIf = auto()
```

```
    cmdElif = auto()
```

```
    cmdElse = auto()
```

```
    funcDecl = auto()
```

```
    funcRtn = auto()
```

```
    semicolon = auto()
```

```
    commaSep = auto()
```

opNot = auto()  
opAnd = auto()  
opOr = auto()  
constTrue = auto()  
constFalse = auto()  
opAtt = auto()  
opEquals = auto()  
opDiff = auto()  
opGtrThan = auto()  
opLessThan = auto()  
opGtrEqual = auto()  
opLessEq = auto()  
opAdd = auto()  
opSub = auto()  
opUnaryNeg = auto()  
opUnaryPos = auto()  
opDiv = auto()  
opMult = auto()  
opMod = auto()  
opConcat = auto()  
funcRead = auto()  
funcPrint = auto()  
paramBegin = auto()  
paramEnd = auto()  
arrayBegin = auto()  
arrayEnd = auto()  
scopeBegin = auto()  
scopeEnd = auto()  
notDefined = auto()  
EOF = auto()

## **Expressões regulares auxiliares:**

letters: [a-zA-z]

digits: [0-9]

specialCharacters: ‘ ‘ | ‘\‘ | ‘\~’ | ‘\!’ | ‘\@’ | ‘\#’ | ‘\\$’ | ‘\%’ | ‘\^’ |  
‘\&’ | ‘\\*’ | ‘(’ | ‘)’ | ‘\.’ | ‘\\_’ | ‘\+’ | ‘\=’ | ‘\’ | ‘\{’ | ‘\}’ | ‘\[’ | ‘\]’ | ‘\’ |  
‘\|’ | ‘\,’ | ‘\.’ | ‘\”’ | ‘\”’ | ‘\?’ | ‘\V’ | ‘\’

## **Expressões regulares:**

identifier: [‘letters’]([‘letters’]digits\\_) +)

## **Ponto inicial de execução do programa:**

main: main

## **Tipos primitivos:**

typeInt: ‘int’

typeFloat: ‘float’

typeBool: ‘bool’

typeChar: ‘char’

typeString: ‘string’

typeVoid: ‘void’

## **Valores que os tipos primitivos podem aceitar:**

intVal: [+ -]? (‘digits’)+

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

boolVal: (‘True’ | ‘False’)    Ok

charVal: (‘\’)(^[a-zA-Z0-9’specialCharacters’]\$)(‘\’)

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

## **Terminais:**

semicolon: ‘;’

## **Comandos para estrutura de repetição:**

cmdFor: ‘for’

cmdWhile: ‘while’

## **Comandos para estrutura condicional:**

cmdIf: ‘if’

cmdElif: ‘elif’

cmdElse: ‘else’

## **Operadores lógicos:**

opNot: ‘!’

opAnd: ‘&&’

opOr: ‘||’

constTrue: ‘True’

constFalse: ‘False’

## **Operadores relacionais:**

opEquals: ‘==’

opGtrThan: ‘>’

opLessThan: ‘<’

opGtrEqual: ‘>=’

opLessEq: ‘<=’

opDiff: ‘!=’

## **Operadores matemáticos:**

opAdd: '+'

opSub: '-'

opUnaryNeg: '-'

opUnaryPos: '+'

opDiv: '/'

opMult: '\*'

opMod: '%'

opAtt: '='

## **Operador de concatenação:**

opConcat: '.'

## **Leitura e escrita:**

funcRead: 'input'

funcPrint: 'print'

## **Funções:**

funcDecl: function

funcRtn: return

## **Separadores:**

commaSep: ','

## **Delimitadores de parâmetros:**

paramBegin: ‘\(  
paramEnd: ‘\)’

### **Arranjos unidimensionais:**

arrayBegin: ‘\['  
arrayEd: ‘\]’

### **Delimitadores de escopo:**

scopeBegin: ‘\{'  
scopeEnd: ‘\}’

### **Tokens não identificados pela linguagem:**

notDefined