

# Analizador Léxico para a Linguagem C- utilizando Máquina de Moore: Projeto de Implementação

Gustavo Zanzin Guerreiro Martins<sup>1</sup>

<sup>1</sup>Departamento Acadêmico de Computação – Universidade Tecnológica Federal do Parná (UTFPR) Caixa Postal 86812-460 – 87301-899 – Campo Mourão – PR – Brazil

gustavozanzin@alunos.utfpr.edu.br

**Abstract.** *This work presents how the implementation of a Lexical Analyzer for the C- language was done. The objective was to implement an automaton with output, of the Moore Machine type that works as a Lexical Analyzer of the C- language. The program written in the Python programming language with the aid of the automata-python library receives as input a source code of the C- language and generates as output a list of tokens that were recognized in the input code.*

**Resumo.** *Este trabalho apresenta como foi feita a implementação de um Analisador Léxico para a linguagem C-. O objetivo foi implementar um automato com saída, do tipo Máquina de Moore que funcione como um Analisador Léxico da linguagem C-. O programa escrito na linguagem de programação Python com o auxílio da biblioteca automata-python recebe como entrada um código-fonte da linguagem C- e gera como saída uma lista de tokens (marcas) que foram reconhecidos no código de entrada.*

## 1. Introdução

O objetivo deste trabalho foi projetar e implementar um autômato com saída, do tipo *Máquina de Moore* que funcione como um Analisador Léxico para a linguagem de programação C-. O programa recebe como entrada um código-fonte na linguagem C- e, como saída, gera uma lista de *tokens*.

## 2. Descrição da Linguagem

Por conta de ser uma versão mais simplificada da linguagem de programação C, a linguagem C- foi a alternativa encontrada para criar um *Analisador Léxico* sem a complexidade que seria a implementação desse mecanismo para reconhecer outras linguagens. A linguagem C- é apresentada no *Apêndice A* do livro *Compiladores: Princípios e Práticas* [LOUDEN 2004] e a sintaxe das suas regras sintáticas são apresentadas no código abaixo.

```
1 program ::= declaration-list
2 declaration-list ::= declaration-list declaration | declaration
3 declaration ::= var-declaration | fun-declaration
4 var-declaration ::= type-specifier ID ; | type-specifier ID [ NUM ] ;
5 type-specifier ::= int | float | void
6 fun-declaration ::= type-specifier ID ( params ) compound-stmt
7 params ::= param-list | void
8 param-list ::= param-list , param | param
9 param ::= type-specifier ID | type-specifier ID [ ]
```

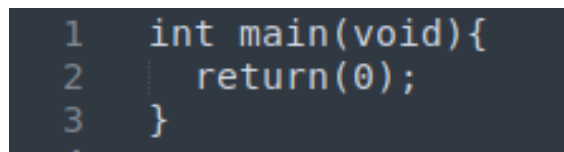
```

10 compound-stmt ::= { local-declarations statement-list }
11 local-declarations ::= local-declarations var-declaration | empty
12 statement-list ::= statement-list statement | empty
13 statement ::= expression-stmt | compound-stmt | selection-stmt |
    iteration-stmt |
14 return-stmt
15 expression-stmt ::= expression ; | ;
16 selection-stmt ::= if ( expression ) statement | if ( expression )
    statement else
17 statement
18 iteration-stmt ::= while ( expression ) statement
19 return-stmt ::= return ; | return expression ;
20 expression ::= var = expression | simple-expression
21 var ::= ID | ID [ expression ]
22 simple-expression ::= additive-expression relop additive-expression |
23 additive-expression
24 relop ::= <= | < | > | >= | == | !=
25 additive-expression ::= additive-expression addop term | term
26 addop ::= + | -
27 term ::= term mulop factor | factor
28 mulop ::= * | /
29 factor ::= ( expression ) | var | call | NUMBER
30 call ::= ID ( args )
31 args ::= arg-list | empty
32 arg-list ::= arg-list , expression | expression

```

### 3. Descrição do Processo de Análise Léxica

O processo de *Análise Léxica* consiste em analisar caractere a caractere de um código-fonte com o intuito de identificar todos os possíveis lexemas de serem formados em uma determinada linguagem de programação para gerar uma lista de *tokens* que têm um significado pré-definido. Alguns exemplos dos *tokens* gerados na saída léxica são: INT, RETURN, VOID, ID, IF e ELSE. Para exemplificar tal procedimento, acompanhe nas Figuras 1 e 2, respectivamente, um simples código em C- e sua lista de tokens gerada após a análise léxica. Além disso, geralmente são usados outros mecanismos, como um analisador, em conjunto ao *lexer* para realizar a análise sintática de um código-fonte.



```

1  int main(void){
2      return(0);
3  }

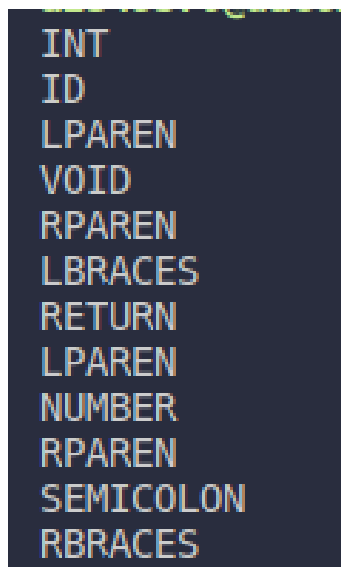
```

Figura 1. Exemplo de código em C-.

### 4. Descrição da Implementação

Para a efetivação prática do *Analisador Léxico*, foi utilizada a linguagem de programação *Python*, com o auxílio da biblioteca *automata-python* para fazer a implementação por meio da *Máquina de Moore*.

O programa recebe um arquivo que contém um código-fonte da linguagem de programação C-, instancia um objeto da classe *Moore* (implementado na



```
INT
ID
LPAREN
VOID
RPAREN
LBRACES
RETURN
LPAREN
NUMBER
RPAREN
SEMICOLON
RBRACES
```

**Figura 2.** Exemplo de lista de *tokens* gerada a partir da do código da Figura 1.

`automata-python`), separa fragmentos do código-fonte para enviar para o autômato realizar a análise léxica e, por fim, imprime os *tokens* reconhecidos.

Além disso, nesta versão alternativa de implementação é utilizado um caractere reservado para indicar quando determinada cadeia sequencial está pronta para ser lida pelo analisador léxico, gerando, assim, os tokens léxicos da maneira correta, sem a ocorrência de problemas como a impressão de tokens ID no decorrer da construção de um INT, por exemplo. Ademais, possibilita que um ID "in", ou seja, um identificador com algumas de suas iniciais iguais as iniciais de um *token* de uma palavra reservada, por exemplo, seja criado.

## 5. Conclusão

Em síntese, mesmo com uma versão de implementação alternativa, foi possível efetuar o funcionamento deste trabalho utilizando a biblioteca `automata-python` da linguagem de programação Python. E, por conseguinte, ampliar os conhecimentos a cerca do funcionamento de automatos com saída, como a Máquina de Moore. Outrossim, foi possível observar e compreender melhor o comportamento do automato de acordo com determinados tipos de entradas diferentes, uma vez que obtivemos sucesso no objetivo inicial de implementar o analisador léxico.

## Referências

LOUDEN, K. C. (2004). *Compiladores: Princípios e Práticas*. São Paulo, SP: Thomson, 1st ed edition.