Analisador Léxico para a Linguagem **C-** utilizando Máquina de Moore

Projeto de Implementação

Prof. Rogério Aparecido Gonçalves[1]

[1] *Universidade Tecnológica Federal do Paraná (UTFPR)*

Departamento de Computação (DACOM)

rogerioag@utfpr.edu.br

11 de maio de 2022

Resumo

Este documento apresenta a especificação do 1º Trabalho de Implementação da disciplina. O objetivo é projetar e implementar um autômato com saída, do tipo *Máquina de Moore* que funcione como um Analisador Léxico para a linguagem **C-**. É esperado que o programa, por linha de comando, receba como entrada um código fonte na linguagem **C-** e gere como saída uma lista de *tokens*. A sintaxe de **C-** com os **tokens** previstos para a linguagem são apresentados, bem como as instruções para o desenvolvimento do trabalho.

Sumário

1 A Linguagem C-

A Linguagem de Programação para a qual iremos implementar um *Analisador Léxico* utilizando *Máquina de Moore* é a linguagem `C-` que é uma simplificação da linguagem `C`. A escolha dessa linguagem foi pelo fato de grande parte das disciplinas introdutórias à programação, como Algoritmos dos cursos de Computação serem ministradas utilizando `C`, assim teríamos a vantagem de ser uma linguagem conhecida pelos alunos. A versão simplificada, `C-` é apresentada no *Apêndice A* do livro texto utilizado na disciplina de Compiladores (LOUDEN 2004) e sua sintaxe tem as regras sintáticas apresentadas no Código 1.

```
program ::= declaration-list
declaration-list ::= declaration-list declaration | declaration
declaration ::= var-declaration | fun-declaration
var-declaration ::= type-specifier ID ; | type-specifier ID [ NUM ] ;
type-specifier ::= int | float | void
fun-declaration ::= type-specifier ID ( params ) compound-stmt
params ::= param-list | void
param-list ::= param-list , param | param
param ::= type-specifier ID | type-specifier ID [ ]
compound-stmt ::= { local-declarations statement-list }
local-declarations ::= local-declarations var-declaration | empty
statement-list ::= statement-list statement | empty
statement ::= expression-stmt | compound-stmt | selection-stmt | iteration-stmt |
    return-stmt
expression-stmt ::= expression ; | ;
selection-stmt ::= if ( expression ) statement | if ( expression ) statement else
    statement
iteration-stmt ::= while ( expression ) statement
return-stmt ::= return ; | return expression ;
expression ::= var = expression | simple-expression
var ::= ID | ID [ expression ]
simple-expression ::= additive-expression relop additive-expression |
    additive-expression
relop ::= <= | < | > | >= | == | !=
additive-expression ::= additive-expression addop term | term
addop ::= + | -
term ::= term mulop factor | factor
mulop ::= * | /
factor ::= ( expression ) | var | call | NUMBER
call ::= ID ( args )
args ::= arg-list | empty
arg-list ::= arg-list , expression | expression
```

Código 1: Regras Sintáticas C-

2 Análise Léxica

A **Análise Léxica** é a fase do compilador que lê o código-fonte do arquivo de entrada como um fluxo de caracteres, e nesse processo de varredura reconhece os *tokens* ou marcas da linguagem. As denominações Sistema de Varredura, Analisador Léxico e *Scanner* são equivalentes.

Devem ser reconhecidas as marcas presentes na linguagem `C-`, como `if`, `while` que são palavras chave ou palavras reservadas. Precisam ser reconhecidos os nomes de variáveis e funções que são os *identificadores*, símbolos e operadores aritméticos, lógicos e relacionais.

O processo de reconhecimento das marcas, a identificação de padrões pode ser feito com a implementação de um analisador léxico utilizando a teoria de *autômatos finitos com saída*, mais especificamente, com uma *Máquina de Moore* gerando como saída cada uma das classes de *tokens*.

Para um código simples em `C-` igual o Código 2, a lista de *tokens* (marcas) que precisam ser identificadas é apresentada no Código 3.

```
/* Programa simples. */

int main(void){
  return(0);
}
```

Código 2: Programa em C-

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

Código 3: Tokens

As classes de *tokens* (marcas) que precisam ser identificadas e que são apresentadas na Tabela 2.

Tabela 2: **Tokens** da linguagem **C-**

palavras reservadas, símbolos, lexemas	Token
if	IF
else	ELSE
int	INT
return	RETURN
void	VOID
while	WHILE
+	PLUS
-	MINUS
*	TIMES
/	DIVIDE
<	LESS
<=	LESS_EQUAL
>	GREATER
>=	GREATER_EQUAL
==	EQUALS
!=	DIFFERENT
(	LPAREN
)	RPAREN
[	LBRACKETS
]	RBRACKETS
{	LBRACES
}	RBRACES
=	ATTRIBUTION
;	SEMICOLON
,	COMMA

Além dessas, as marcas para **identificadores**, como nomes de variáveis e funções, (`ID`) e **números** (`NUMBER`).

2.1 Linguagens de programação para a implementação

Para a implementação do compilador, pode ser utilizado qualquer linguagem de programação. Como sugestão pode ser utilizada a linguagem **Python**, podendo ser utilizadas bibliotecas que auxiliam na implementação de Máquinas de Estados e Autômatos, como automata-lib.

2.2 Testes

Alguns casos de testes estão disponíveis no moodle institucional junto com essa especificação. Serão executados esses e outros testes que o professor julgar necessário durante a avaliação do trabalho.

2.3 Instruções Gerais

1. Faça download do arquivo do modelo de estrutura do trabalho e relatório disponível na página da disciplina no moodle. Descompacte e trabalhe nos arquivos e estrutura fornecida, pois será a mesma estrutura que deverá ser entregue ao final do projeto.
2. O relatório deve ter a descrição do trabalho e da implementação, exemplos de entrada e saída gerada pelo Analisador Léxico.
3. Deverão ser entregues:
   a) O código fonte da implementação.
   b) Relatório em **pdf**.
4. O projeto deve seguir a estrutura de diretórios e arquivos, disponível no formato. A estrutura do projeto é apresentada na Figura 1. **Siga a estrutura fornecida para desenvolver o trabalho.**

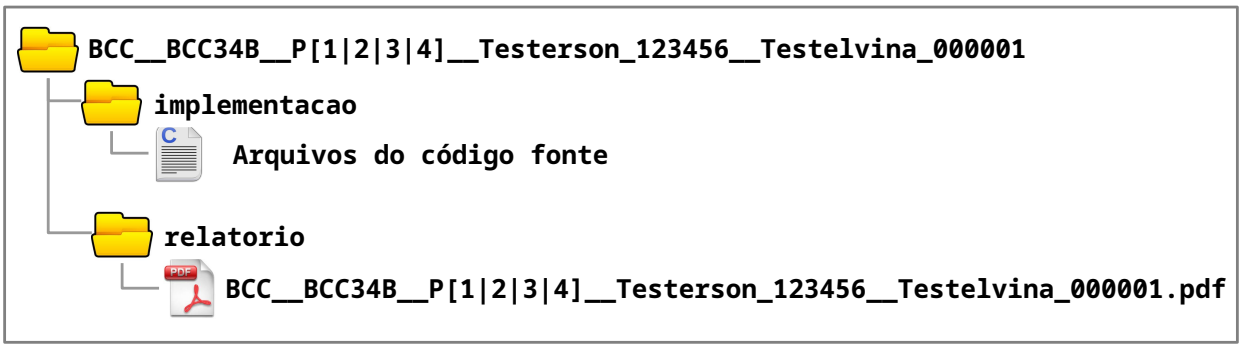

Figura 1: Formato de Entrega

2.4 Relatório

O relatório deve ser feito no formato do `LibreOffice` ou no `Latex`. Utilize o formato para publicação de artigos da Sociedade Brasileira de Computação (SBC)[1][2].

[1]Formato de artigo da SBC: http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros

[2]Formato de artigo da SBC no overleaf: https://www.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr

2.5 Avaliação

Será avaliado o funcionamento do analisar léxico para a linguagem de programação `c-`.

- O programa deve receber como entrada um arquivo com um código fonte em `c-` pela linha de comando.
- A **saída** será uma lista de *tokens* identificados.
- Serão avaliados, dentre outros critérios: **a) Da implementação:** + O funcionamento do programa. + O capricho e a organização na elaboração do projeto. + A corretude da implementação em relação ao que foi pedido no trabalho. + A colocação em prática dos conceitos que foram discutidos em sala de aula de forma correta. + A qualidade do projeto e da implementação (descrição e elaboração do projeto e o passo a passo da implementação).

  b) Do relatório: + O conteúdo e a forma que foi apresentado, se o formato é o mesmo solicitado. + Organização das ideias e do desenvolvimento. + O capricho na elaboração e na formatação do texto, bem como o conteúdo do texto.
- Não serão avaliados os trabalhos:

  a) Que chegarem fora do prazo.
  b) Que não forem feitos no formato especificado.
  c) Que não foram compactados em um só arquivo.
  d) Que não tiverem identificação (nome e matrícula).
  e) Que forem cópias de outros trabalhos ou materiais da internet.
  f) Que não seguirem todas estas instruções.
- Não se esqueça que o trabalho vale **10,0** e contribui para o cálculo da nota final.

2.6 Entrega e Apresentação

O trabalho será **individual** e deverá ser entregue até o dia **30/05/2022** no moodle da disciplina em um pacote compactado. A estrutura do projeto com os arquivos do projeto (fonte e relatório) deve ser compactada (zipados) e o arquivo compactado deve ser enviado pelo moodle utilizando a opção de submissão **"1º Trabalho"**, o nome do arquivo compactado deve seguir o padrão de nomes do formato.

Deverá ser especificado na entrega o mecanismo de execução do programa para a realização da correção.

Obs.: Favor utilizar ZIP como forma de compactação. O Relatório deve ser entregue impresso, no horário da aula, para o professor.

Referências

LOUDEN, Kenneth C. 2004. *Compiladores: Princípios e Práticas.* 1st ed. São Paulo, SP: Thomson.

Menezes, Paulo Blauth. 2011. *Linguagens Formais e Autômatos.* Bookman. https://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000000444&lang=pt-br&site=eds-live&scope=site.

Sipser, Michael. 2007. *Introdução à Teoria Da Computação.* Cengage Learning. https://search.ebscohost.com/login.aspx?direct=true&db=edsmib&AN=edsmib.000008725&lang=pt-br&site=eds-live&scope=site.