

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

Hudson Taylor P. Cassim<sup>1</sup>

Departamento Acadêmico de Computação (DACOM)  
Universidade Tecnológica Federal do Paraná (UTFPR)

## Abstract

Lexical analysis is a fundamental step in the code compilation process. Its main purpose is to interpret the source code of a program, processing the input file as a stream of characters, and transforming it into a sequence of tokens (meaningful language units). This article presents the design and implementation of a lexical analyzer for the C- language, using the concept of finite automata, specifically the model known as Moore's Machine.

## Resumo

A análise léxica é uma etapa fundamental no processo de compilação de um código. Seu principal objetivo é interpretar o código-fonte de um programa, processando o arquivo de entrada como um fluxo de caracteres e transformando-o em uma sequência de tokens (unidades significativas da linguagem). Este artigo apresenta o projeto e a implementação de um analisador léxico para a linguagem C-, utilizando o conceito de autômatos finitos, especificamente o modelo conhecido como Máquina de Moore.

## 1 Introdução

Este trabalho tem como objetivo desenvolver um analisador léxico para a linguagem C- utilizando o conceito de Máquina de Moore. O processo foi dividido em duas fases principais, descritas ao longo deste documento:

- **Projeção do Autômato:** Nesta etapa, foi utilizado o software Java Formal Language and Automata Package (JFLAP) para criar o diagrama do autômato responsável por reconhecer os *tokens* da linguagem. Este software é uma ferramenta interativa de visualização e ensino para linguagens formais que permite aos usuários criarem e operarem autômatos, gramáticas e expressões regulares (Susan H. Rodge 2006).
- **Implementação do Analisador Léxico:** A segunda fase consistiu na implementação do código do analisador léxico, desenvolvida em *Python*. Essa escolha se deve à ampla disponibilidade de bibliotecas que facilitam a modelagem de autômatos, como a *automata-lib*. Essa biblioteca, gratuita e de código aberto (Licença Apache, v2.0), permite a modelagem de autômatos, gráficos e sistemas de transição de maneira eficiente (Frohme 2017). Nesta etapa também são realizados testes automatizados para verificação de erros no sistema.

Este documento também aborda tópicos essenciais para a compreensão do trabalho, incluindo uma visão geral da linguagem C-, o papel da análise léxica no processo de compilação e exemplos práticos de entradas e saídas processadas pelo analisador desenvolvido. Esses conceitos são fundamentais para contextualizar e esclarecer os aspectos abordados no projeto.

## 2 A linguagem C-

A linguagem C- foi escolhida como base para o desenvolvimento do analisador léxico. Derivada da linguagem C, ela foi projetada especificamente para ensinar os fundamentos do desenvolvimento de compiladores, conforme proposto pelo professor LOUDEN (2004). Ao simplificar as complexidades presentes na linguagem C, a C- oferece uma abordagem mais acessível, ideal para explorar e implementar os processos envolvidos na criação de um compilador, incluindo a etapa de análise léxica.

Esta linguagem apresenta características semelhantes às da linguagem C, porém de forma simplificada, o que facilita o entendimento e o desenvolvimento do projeto, especialmente no que diz respeito aos aspectos semânticos. Algumas das principais características da linguagem C- estão listadas abaixo:

- Conjunto reduzido de tipos de dados: Nesta linguagem são aceitos apenas 2 tipos de dados, sendo eles o tipo `int` e o tipo `void`. Este conjunto é simples comparado ao de outras linguagens, como o caso da linguagem C onde há uma ampla gama de tipagens como o `float`, `char`, `double` etc.
- Estruturas de controle básicas: A linguagem C- aceita apenas dois tipos de estruturas de controles em seu código. Estas estruturas são os condicionais *if/else* e laços de repetição com a estrutura *while*.
- Funções: A linguagem C- tem a capacidade para criação de funções, porém estas devem seguir o conjunto reduzido de tipagem de dados, assim sendo os parâmetros de entrada e retornos da função necessariamente devem do tipo *int* ou *void*.
- Entrada e Saída: Comandos de entradas e saídas como *printf*, *scanf*, *fread* entre outros, não são reconhecidos pela linguagem C-.

O código 1 apresenta um exemplo de uma função de soma, escrita na linguagem C-. Observe-se sua semelhança e compatibilidade com a linguagem C, o que torna seu uso mais intuitivo para aqueles que já possuem conhecimento em C. Essa familiaridade facilita a compreensão e aplicação da linguagem C-, especialmente por se tratar de uma versão simplificada.

Código 1: Código em C-

```
1 int sum(int n) {  
2     int i;  
3     int total;  
4     total = 0;  
5     i = 1;  
6     while (i <= n) {  
7         total = total + i;  
8         i = i + 1;  
9     }  
10    return total;  
11 }
```

As palavras reservadas, símbolos e lexemas utilizados no passo da análise léxica da linguagem C- são demonstrado na figura 1 com os seus respectivos *tokens*.

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

Figura 1: Tokens da linguagem C-

Por fim podemos concluir que a linguagem C- é, uma ferramenta didática poderosa, projetada para simplificar o aprendizado de conceitos complexos, enquanto mantém a relevância prática para a construção de compiladores reais.

### 3 O Processo de Análise Léxica

Para compreender o que é a análise léxica, também chamada de sistema de varredura, é fundamental ter uma noção básica do funcionamento de um compilador e do papel que a fase de análise léxica desempenha dentro deste.

Segundo LOUDEN (2004), um compilador é um programa responsável por traduzir um código escrito em uma linguagem-fonte para outra linguagem-alvo. Esse processo ocorre, na maioria dos casos, para converter códigos de linguagens de alto nível em código de máquina, ou seja, de baixo nível.

Este processo de compilação pode ser dividido em alguns passos, conforme a seguinte lista:

1. Analisador léxico ou Sistema de varredura
2. Analisador sintático
3. Analisador semântico
4. Otimizador de código-fonte
5. Gerador de código
6. Otimizador de código-alvo

Dessa forma, nota-se que a análise léxica é a primeira fase do processo de compilação. Nessa etapa, o código-fonte é recebido como entrada e, após o processamento, é convertido em uma sequência de *tokens*, que representam as unidades léxicas da linguagem. Esses *tokens* gerados são então repassados às fases subsequentes do compilador (LOUDEN 2004).

Os *tokens* representam as unidades léxicas da linguagem e são a saída gerada pelo analisador léxico. Eles servem como entrada para as etapas seguintes do processo de compilação. Cada linguagem possui seu próprio conjunto de *tokens*, ou marcas, que definem sua estrutura. Conforme mencionado na Seção 2 os tokens da linguagem C- estão ilustrados na Figura 1.

Ainda, segundo LOUDEN (2004) um analisador léxico processa a entrada como uma sequência de caracteres e itera sobre ela, identificando padrões. Quando um conjunto de caracteres corresponde a um token válido da linguagem, esse token é gerado como saída do processo de análise léxica.

Para identificar esses padrões, um analisador léxico geralmente utiliza uma das duas abordagens principais: expressões regulares ou autômatos finitos. Neste trabalho, é abordado o uso de autômatos finitos, uma vez que a linguagem C- é reduzida e pode ser modelada de forma simplificada, com maior facilidade do que uma linguagem complexa. Especificamente, foi adotado o modelo de Máquina de Moore para a implementação do analisador.

### 3.1 Autômatos Finitos

Os autômatos finitos são modelos simplificados de computadores de forma a reduzir a complexidade de um computador normal (SIPSER 2007). Um autômato finito tem este nome, pois é um modelo computacional que possui um número limitado e fixo de estados, diferentemente dos autômatos infinitos.

Estes estados possuem 3 tipos: estado inicial, ou seja estado em que o processo inicia a leitura; estado de aceitação, este é o estado final, se ao fim do processamento a máquina estiver neste estado significa que a entrada é aceita; estado de não aceitação: são todos os demais estados de transição. O autômato processa uma entrada símbolo por símbolo e muda de estado de acordo com uma função de transição predefinida. Segundo SOUSA (2021) um autômato finito é composto basicamente de 3 partes:

- Fita: Informações de entrada ( O código em C- no caso deste trabalho).
- Unidade de controle: Mecanismo que faz a leitura da fita entrada por entrada.
- Função de transição: Ao ler uma entrada da fita, através da unidade de controle a função de transição leva para o próximo estado da máquina.

Os autômatos finitos podem ser divididos em duas áreas:

- Autômatos Finitos Determinísticos (AFD) : Neste tipo de autômato, cada entrada da fita leva a um somente um estado definido, e cada estado deve conter saídas para todas as entradas possíveis SOUSA (2021).
- Autômatos Finitos Não Determinísticos (AFN) : Este tipo de autômato permite a partir de uma mesma entrada da fita chegar a um conjuntos de próximos estados, por isso o nome não determinístico SOUSA (2021).

Neste trabalho a ênfase será dada aos autômatos finitos determinísticos, visto que a máquina de Moore é uma implementação desse modelo de autômato. A figura 2 demonstra um exemplo de AFD.

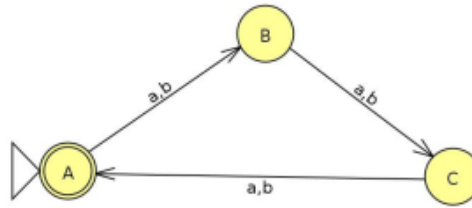


Figura 2: Exemplo de AFD.

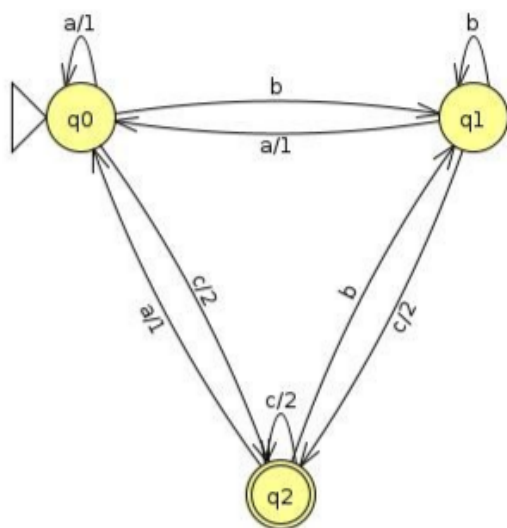
Os AFD também são definidos pela 5-upla:  $M = (\Sigma, Q, \delta, q_0, F)$ .

- $\Sigma$  : Alfabeto de entrada.
- $Q$  : Conjunto de estados possíveis do autômato.
- $\delta$  : Função de transição.
- $q_0$  : Estado inicial.
- $F$  : Conjunto de estados finais.

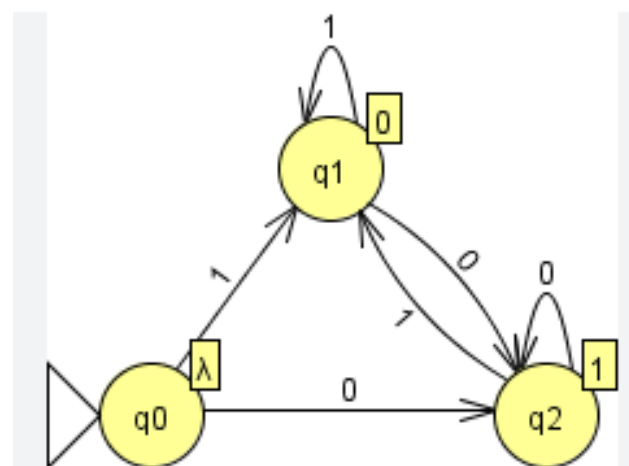
### 3.2 Autômatos finitos com saída e máquina de Moore

Automâtos finitos têm as suas restrições quanto à saída de determinadas entradas, estas restrições seguem 2 condições: aceita ou não aceita. Porém, sem alterar essas características necessárias de um AFD, é possível estender a definição de um AFD, incluindo-se a geração de uma palavra de saída. Esta característica é a base dos autômatos finitos com saída.

Essas saídas podem estar relacionadas às transições o que representa uma máquina de Mealy, conforme figura 3a, ou quanto aos estados, conforme figura 3b esta representa uma máquina de Moore.



(a) Mealy.



(b) Moore.

Figura 3: Autômatos finitos com saída.

Esta saída gerada não pode ser lida como uma nova entrada, ou seja, ser usada como memória, assim como um AFD possui um alfabeto de entrada, um autômato finito com saída possui um alfabeto de saída. A saída é inserida em uma fita separada da fita de entrada, a cada estado ou

transição a fita é escrita e a unidade de controle da fita de saída é movida uma posição para a direita.

A máquina de Moore é definida pela 7-upla:  $M = (\Sigma, Q, \delta, q_0, F, \Delta, \delta_S)$ .

- $\Sigma$  : Alfabeto de entrada.
- $Q$  : Conjunto de estados possíveis do autômato.
- $\delta$  : Função de transição.
- $q_0$  : Estado inicial.
- $F$  : Conjunto de estados finais.
- $\Delta$  : Alfabeto de saída
- $\delta_S$  : Função de saída.

Observa-se as mesmas características de um AFD, porém possui duas definições a mais: o alfabeto de saída e a função de saída. A cada estado, um elemento do alfabeto de saída é escrito na fita de saída, podendo ser escrita também a palavra vazia. Ao final, tem-se o resultado de aceito/não aceito de um AFD, junto com a saída da fita.

Dessa forma, a máquina de Moore é utilizada para implementar um analisador léxico. A cada entrada do código (fita) a máquina se dirige a um estado, nesse estado caso seja encontrado um token da linguagem este token é gravado na fita de saída.

## 4 Implementação

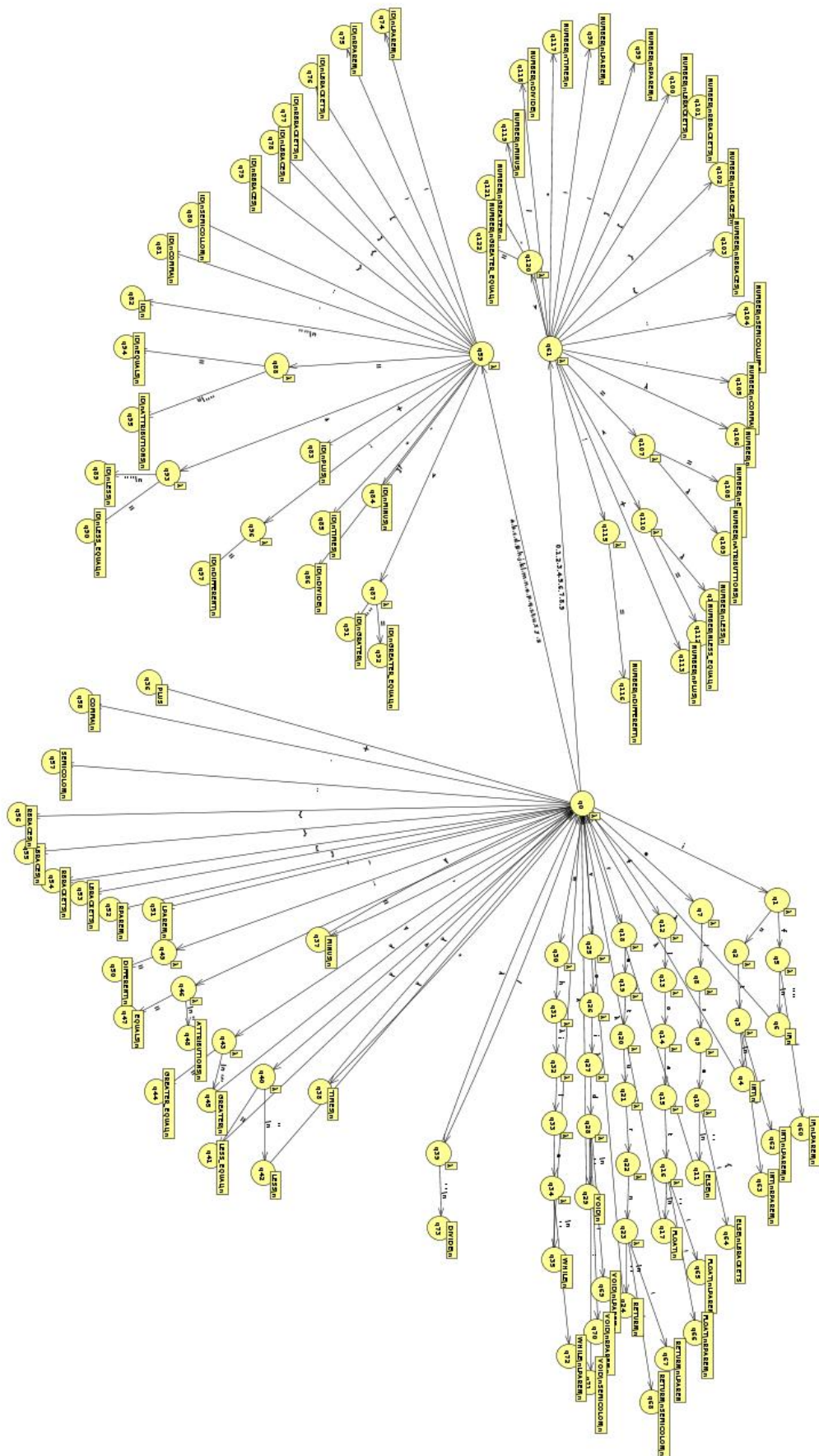
Conforme o texto da seção 1 a implementação é dividida em duas etapas: a projeção do trabalho e a implantação do analisador léxico.

### 4.1 Projeção

Para realizar a projeção do trabalho foi utilizado a ferramenta do JFLAP. Esta ferramenta conta com a opção de desenvolver projeto do tipo Moore Machine, utilizada neste trabalho. Desta forma, para realizar a projeção é necessário colocar todos os estados possíveis e realizar os relacionamentos entre eles.

Embora a linguagem C- seja uma linguagem reduzida a parte referente a projeção ainda ficou com um tamanho deveras grande, visto que foi necessário a criação de 132 estados e mais de 1500 transições. Esta etapa foi um processo trabalhoso, pois no autômato determinístico cada estado deve ter as suas transições para todas as entradas do alfabeto de entrada, dessa forma para realizar isto no JFLAP é necessário um certo tempo.

Porém depois de um projeto bem realizado a parte do código ganha um certo quesito de facilidade, visto que, para desenvolver o código tem-se o projeto como base de desenvolvimento. O Projeto pode ser observado na figura 4.



## 4.2 Implementação do código

Para implementação do código do projeto foi tomado por base o código inicial disponibilizado pelo professor na linguagem Python utilizando a biblioteca automata-lib. O código utiliza a classe Moore dessa biblioteca segundo o documento oficial da biblioteca automata-lib esta classe tem a estrutura demonstrada no código 2.

Código 2: Moore function

```
1 class Moore(  
2     states: list[str],  
3     input_alphabet: list[str],  
4     output_alphabet: list[str],  
5     transitions: dict[str, dict[str, str]],  
6     initial_state: str,  
7     output_table: dict[str, str]  
8 )
```

Observa-se então que a classe é inicializada com uma lista de estados, o alfabeto de entrada, o alfabeto de saída, um dicionário com as transições entre os estados, o estado inicial e por fim um dicionário com as saídas escritas por cada estado.

Dessa forma, os próximos passos foram inicializar o autômato colocando no código as informações disponíveis através do projeto realizado no JFLAP.

Após esse processo tem-se a máquina de Moore inicializada, e agora pronta para receber um código na linguagem C- como entrada. Este processo é feito na main. Nessa etapa é verificado os parâmetros informado na execução do código. Os parâmetros verificados são o parâmetro -k que é opcional e sua função é que caso seja informado apresentará apenas os tokens gerados pela máquina de moore, e caso não seja informado exibirá um detalhamento completo do autômato, o próximo parâmetro é um arquivo .cm contendo o código na linguagem C-, este arquivo então é passado para o método *moore.get\_output\_from\_string(<arquivo>)* que retornará os tokens gerados pela máquina de Moore. A função main pode ser vista no código 3 e um exemplo do comando para compilação seria esse: *<python analex.py -k testeC-.cm>*

Foi disponibilizado também pelo professor uma classe de teste para verificar o funcionamento do código, estes testes contém alguns códigos escritos em C- e a saída que se espera de tokens gerados pelo autômato, estes testes são realizados a cada commit, mas também podem ser realizados utilizando o comando: *pytest -v*.



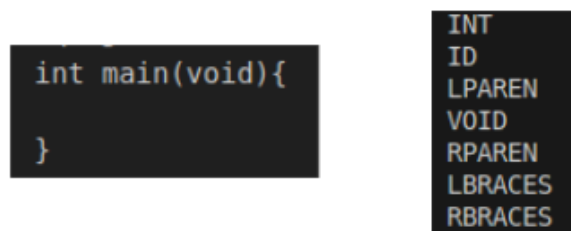
Código 3: Main function

```
1 def main():
2     check_cm = False
3     check_key = False
4
5     for idx, arg in enumerate(sys.argv):
6         #print("Argument #{0} is {1}".format(idx, arg))
7         aux = arg.split('.')
8         if aux[-1] == 'cm':
9             check_cm = True
10            idx_cm = idx
11
12            if(arg == "-k"):
13                check_key = True
14
15    #print("No. of arguments passed is ", len(sys.argv))
16
17    if(len(sys.argv) < 3):
18        raise TypeError(error_handler.newError(check_key, 'ERR-LEX-USE'))
19
20    if not check_cm:
21        raise IOError(error_handler.newError(check_key, 'ERR-LEX-NOT-CM'))
22    elif not os.path.exists(sys.argv[idx_cm]):
23        raise IOError(error_handler.newError(check_key, 'ERR-LEX-FILE-NOT-EXISTS'))
24
25    else:
26        data = open(sys.argv[idx_cm])
27        source_file = data.read()
28
29        if not check_cm:
30            print("Def")
31            print(moore)
32            print("Entrada:")
33            print(source_file)
34            print("Lista de Tokens:")
35
36        print(moore.get_output_from_string(source_file))
```

### 4.3 Exemplos de execução do código

Esta subseção tem por objetivo demonstrar alguns exemplos do funcionamento do código.

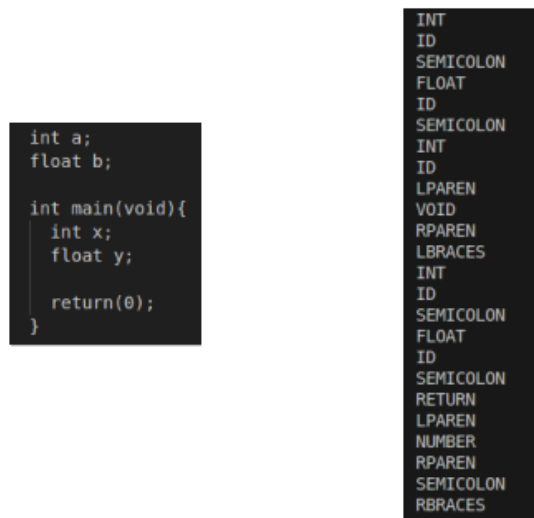
#### 4.3.1 Exemplo 1



The figure consists of two side-by-side terminal screenshots. The left screenshot shows a C function signature: `int main(void){` on the first line and `}` on the second line. The right screenshot shows the output of a tokenization process, listing tokens one per line: `INT`, `ID`, `LPAREN`, `VOID`, `RPAREN`, `LBRACES`, and `RBRACES`.

Figura 5: Exemplo 1.

### 4.3.2 Exemplo 2



```
int a;
float b;

int main(void){
    int x;
    float y;

    return(0);
}
```

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

Figura 6: Exemplo 2.

## 5 Conclusão

O trabalho apresentou o desenvolvimento de um analisador léxico para a linguagem C-, utilizando o modelo de Máquina de Moore. A implementação foi dividida em duas etapas principais: a projeção do autômato finito determinístico (AFD) utilizando o JFLAP e a codificação do analisador em Python, com suporte da biblioteca automata-lib.

Por fim, os testes automatizados garantiram a confiabilidade do analisador, demonstrando que ele pode ser utilizado para fins acadêmicos e para a compreensão do processo de análise léxica dentro de um compilador. Assim, o trabalho reforça a importância dos autômatos finitos e das máquinas de Moore no estudo da Teoria da Computação e na construção de ferramentas para processamento de linguagens formais.

## Referências

- Frohme, M. 2017. automatalib. <https://github.com/LearnLib/automatalib>.
- LOUDEN, Kenneth C. 2004. *Compiladores: Princípios e práticas*. São Paulo, SP: Thomson 1st edn.
- SIPSER, Michael. 2007. *Introdução à teoria da computação: Trad. 2ª ed. norte-americana*. Porto Alegre, RS: +A Educação - Cengage Learning Brasil 1st edn.
- SOUSA, Leonardo B G.; MARTINS Rafael L.; et al., Carlos E B.; NASCIMENTO. 2021. *Linguagens formais e autômatos*. Porto Alegre, RS: SAGAH 1st edn.
- Susan H. Rodge, Thomas W. Finley. 2006. *Jflap – an interactive formal languages and automata package*. São Sudbury, MA: Jones Bartlett Publishers.