

**UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

RELATÓRIO SOBRE A CONSTRUÇÃO DO ANALISADOR LÉXICO

Yuri Luis Malinski Lanzini

RESUMO

Este trabalho apresenta a implementação de um analisador léxico voltado para o reconhecimento de tokens em linguagens de programação. O projeto inclui uma estrutura baseada em funções de estado explícitas para reconhecer palavras-chave, operadores, separadores, números e identificadores, além de reportar erros encontrados no código-fonte. A abordagem utilizada emprega transições de estado baseadas em autômatos finitos determinísticos (AFD). São discutidos também aspectos da criação da tabela de símbolos e da geração de uma fita de saída para a identificação de tokens ou erros. O resultado final é um analisador funcional que pode ser utilizado nas etapas subsequentes de compilação, como a análise sintática e semântica.

1 INTRODUÇÃO

Compiladores são ferramentas fundamentais no desenvolvimento de software, permitindo que linguagens de programação de alto nível sejam traduzidas para código de máquina executável. Entre as principais etapas de um compilador está a análise léxica, responsável por reconhecer e categorizar tokens no código-fonte. Esta etapa é crucial, pois define a base para a análise sintática e semântica subsequente.

O objetivo deste trabalho é implementar um analisador léxico para uma linguagem simples, utilizando técnicas de AFD. O analisador reconhece palavras-chave como `if`, `else` e `return`, operadores aritméticos e relacionais, números inteiros e de ponto flutuante, além de identificadores. O problema a ser resolvido envolve a criação de um mecanismo eficiente que seja capaz de processar o código-fonte e identificar tokens corretamente, gerando uma fita de saída e uma tabela de símbolos.

2 REFERENCIAL TEÓRICO

A análise léxica é a primeira fase de um compilador, onde a sequência de caracteres de entrada é transformada em uma sequência de tokens. Um token pode ser uma palavra-chave, operador, separador ou identificador. Para implementar um analisador léxico, utiliza-se um autômato finito determinístico (AFD), que processa os caracteres de entrada e transita entre estados com base em regras pré-definidas.

Um AFD consiste em um conjunto de estados, um estado inicial, um conjunto de estados finais e um conjunto de transições que definem como o analisador move-se entre estados ao ler cada caractere. Um estado de erro também é necessário para tratar entradas inválidas. O analisador léxico proposto neste trabalho adota essa técnica para reconhecer tokens.

3 IMPLEMENTAÇÃO E RESULTADOS

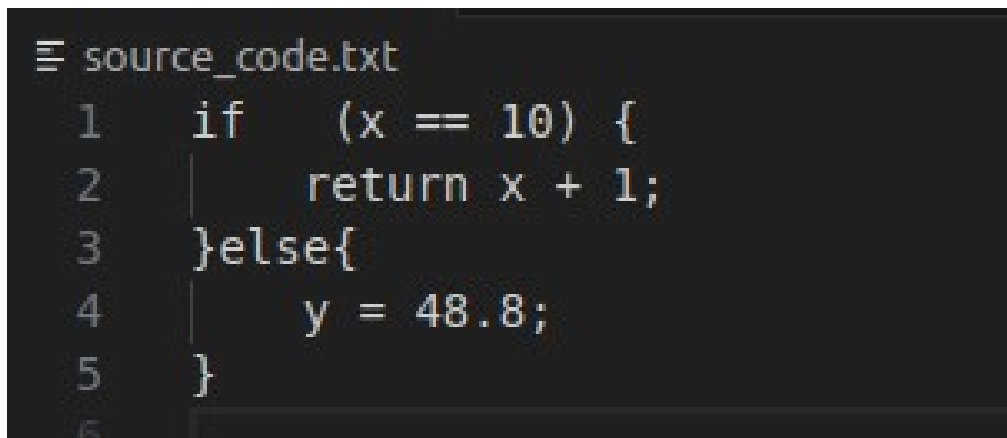
A implementação do analisador léxico foi feita utilizando a linguagem Python. A seguir, apresenta-se o processo de implementação e os resultados obtidos:

Especificação:

A classe `LexicalAnalyzer` foi criada para processar o código-fonte e gerar a fita de saída e a tabela de símbolos. Cada estado do autômato é representado por uma função que processa os caracteres lidos e decide as transições de estado. As principais funcionalidades incluem:

- **Leitura do Código-Fonte:** O código-fonte é lido de um arquivo e armazenado para ser processado.
- **Estados para Tokens Específicos:** Funções como `_q1`, `_q2`, etc., representam os estados do autômato e são responsáveis por reconhecer tokens como palavras-chave, operadores e separadores.
- **Tabela de Símbolos:** Identificadores e seus valores são armazenados em uma tabela de símbolos, que é usada para rastrear as variáveis e constantes presentes no código.

Exemplo de execução ao processar o seguinte trecho de código:

A screenshot of a code editor with a dark background. The file name 'source_code.txt' is visible at the top left. The code is a Python snippet with line numbers 1 through 6 on the left margin. The code itself is: 1 if (x == 10) {, 2 | return x + 1;, 3 }else{, 4 | y = 48.8;, 5 }, 6 |. The code is color-coded: 'if' is blue, 'return' is red, 'else' is blue, and 'y' is green.

```
≡ source_code.txt
1  if (x == 10) {
2  |     return x + 1;
3  }else{
4  |     y = 48.8;
5  }
6  |
```

O analisador léxico gerou a seguinte fita de saída e tabela de símbolos:

```
Fita de Saída:
KEYWORD(if)
SEPARATOR(open_parenthesis)
ID(x)
OPERATOR(==)
INTEGER(10)
SEPARATOR(close_parenthesis)
SEPARATOR(open_brace)
KEYWORD(return)
ID(x)
OPERATOR(+)
INTEGER(1)
SEPARATOR(semicololon)
SEPARATOR(close_brace)
KEYWORD(else)
SEPARATOR(open_brace)
ID(y)
OPERATOR(=)
FLOAT(48.8)
SEPARATOR(semicololon)
SEPARATOR(close_brace)
$

Tabela de Símbolos:
x: {'type': 'identifier', 'value': None, 'value_type': None}
y: {'type': 'identifier', 'value': 48.8, 'value_type': 'FLOAT'}
```

Validação:

A implementação foi validada com exemplos de código-fonte contendo palavras-chave, operadores e identificadores. O analisador conseguiu identificar corretamente os tokens, mesmo em situações onde erros foram simulados (como números com múltiplos pontos ou caracteres inválidos).

4 CONSIDERAÇÕES FINAIS

O projeto implementou com sucesso um analisador léxico funcional para uma linguagem de programação simples. O código desenvolvido foi capaz de reconhecer palavras-chave, operadores, números e identificadores, além de gerar uma tabela de símbolos para auxiliar nas fases subsequentes de um compilador.

Entre as dificuldades encontradas, destacam-se a necessidade de tratamento adequado de erros léxicos e a manipulação de diferentes tipos de números. No entanto, esses desafios foram superados com a implementação de um estado de erro e uma lógica de controle mais robusta para números flutuantes.

O resultado final demonstra a importância de uma análise léxica eficiente como parte integrante do processo de compilação, evidenciando a relevância deste componente na construção de ferramentas de software mais complexas.