Problema 1: Analisador Léxico

Igor F. Soares[1]

[1] UEFS – Universidade Estadual de Feira de Santana
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

ifs5544@gmail.com

Abstract. *This report details the implementation of a lexical analyzer in Python for the EXA869 course, which converts source code into tokens using finite automata. The system identifies various language elements such as identifiers, numbers, strings, characters, delimiters, comments, and operators through specific automata. It includes the development of a graphical user interface to display the source code, tokens, errors, and symbol table. Tests confirmed the analyzer's effectiveness in detecting valid lexemes and lexical errors. The project demonstrates a robust and modular solution for lexical analysis.*

Resumo. *Este relatório detalha a implementação de um analisador léxico em Python para a disciplina EXA869, que converte o código-fonte em tokens usando autômatos finitos. O sistema identifica diversos elementos da linguagem, como identificadores, números, strings, caracteres, delimitadores, comentários e operadores, por meio de autômatos específicos. Inclui a construção de uma interface gráfica para exibir o código-fonte, tokens, erros e a tabela de símbolos. Testes confirmaram a eficácia do analisador na detecção de lexemas válidos e erros léxicos. O projeto demonstra uma solução robusta e modular para a análise léxica.*

1. Introdução

A análise léxica é a etapa inicial no processo de transformação do código. Nessa fase, a sequência de caracteres do código-fonte é convertida em tokens, como palavras reservadas da linguagem, constantes e identificadores. O responsável por essa conversão é o analisador léxico, que faz parte do compilador e executa essa função [Ricarte 2003].

O presente relatório documenta a implementação de um analisador léxico solicitado no Problema 1 da disciplina EXA869 - MI Processadores de Linguagens de Programação.

2. Materiais e Métodos

O analisador léxico foi implementado utilizando a linguagem de programação Python, na versão 3.8.1. Foram utilizados apenas pacotes nativos da linguagem, tais como, "*tkinter*" e "*os*". O "*tkinter*" é o pacote padrão do Python para a implementação de interfaces gráficas. Em geral, o pacote é instalado durante a instalação do interpretador Python na máquina. Já o "*os*" é um módulo que permite utilizar ferramentas do sistema operacional, como a verificação de em qual sistema operacional (SO) a aplicação Python está sendo executada, criação de pastas, acesso a arquivos etc.

O sistema foi desenvolvido com base na abordagem de autômatos finitos. Um autômato finito é composto por um conjunto de estados. O estado inicial determina o começo da identificação da palavra. O primeiro elemento da palavra entra no estado inicial do autômato, ao passo que cada um dos elementos do lexema são lidos, a máquina transita de um estado a outro, de acordo com o conjunto de regras de transição estabelecidas para o autômato. Caso ao final da palavra o autômato se encontre em um estado dito final, o lexema é aceito, ou seja, pertence a linguagem. Um autômato é descrito por uma quíntupla do tipo: M = (K, Σ, δ, s, F), onde K é o conjunto finitos de estados; Σ é o alfabeto de entrada; δ o conjunto de transições de estado; s o estado inicial; e F o conjunto de estados finais possíveis [Hopcroft 2002].

Para implementação do sistema, buscando modularizar e simplificar a sua implementação, foram elaborados diferentes autômatos, de modo que cada um dos autômatos tivesse a função de reconhecer diferentes elementos dentro da linguagem. Foi desenvolvido um autômato para a identificação de *Identifiers* (que também é utilizado para a identificação de *Keywords*), um autômato para identificação de *Numbers*, um autômato para identificação de *String*, um autômato para identificação de *Character*, um autômato para *Delimiters*, um autômato para comentários em bloco (*block comment*), um autômato para comentários de linha única (*unique line comment*) e um conjunto de autômatos para identificação dos *Operators*.

Foram construídos diagramas de transição para cada um dos autômatos elaborados. Na Figura 1, pode ser visto o diagrama de transição do autômato responsável pela identificação de *Identifiers*. O autômato é composto por dois estados, sendo apenas um final. Além dos identificadores, esse autômato também é capaz de reconhecer as palavras-chaves, neste caso, se o lexema reconhecido pelo autômato for composto apenas por letras (a..z), é verificado se ele faz parte do conjunto de palavras reservadas, se fizer, é atribuído a *Keyword*, caso contrário é um identificador.

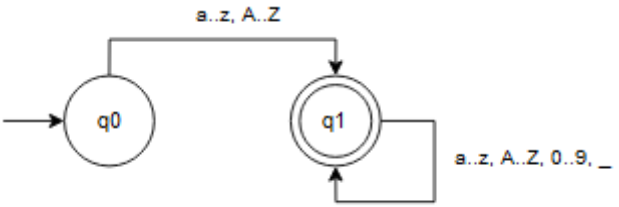

Figura 1. Diagrama de transição do autômato para *Identifiers*

A formalização do autômato *Identifiers* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1};

Σ = {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, _};

δ = {(q0, a..z) = q1; (q0, A..Z) = q1; (q1, a..z) = q1; (q1, A..Z) = q1; (q1, 0..9) = q1; (q1, _) = q1 };

s = q0;

F = {q1}.

Na Figura 2, é exibido o diagrama de transição para o autômato responsável por identificar os *Numbers*. Este autômato atua na identificação de números inteiros e de ponto flutuante. O autômato é composto por quatro estados, sendo dois finais.

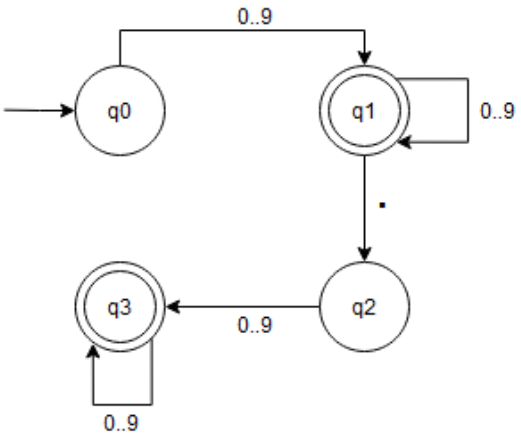

Figura 2. Diagrama de transição do autômato para *Numbers*

A formalização do autômato *Numbers* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2, q3};

Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .};

δ = {(q0, 0..9) = q1; (q1, 0..9) = q1; (q1, .) = q2; (q2, 0..9) = q3; (q3, 0..9) = q3};

s = q0;

F = {q1, q3}.

Já na Figura 3, é exibido o diagrama de transição para a identificação de *String*. O autômato é composto por três estados, sendo apenas um final.

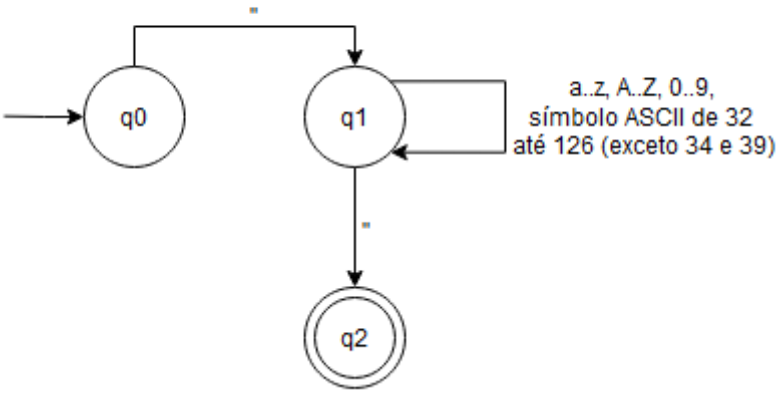

Figura 3. Diagrama de transição do autômato para *String*

A formalização do autômato *String* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2};

Σ = {!, **"**, #, $,%,&, (,), *, +, ,, -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,v, w, x, y, z, {, | , } , ~,};

δ = {(q0, ") = q1; (q1, Σ exceto {"}) = q1; (q1, ") = q2};

s = q0;

F = {q2}.

O diagrama de transição do autômato para a identificação de *Character* é exibido na Figura 4. O autômato é composto por quatro estados, sendo apenas um final.

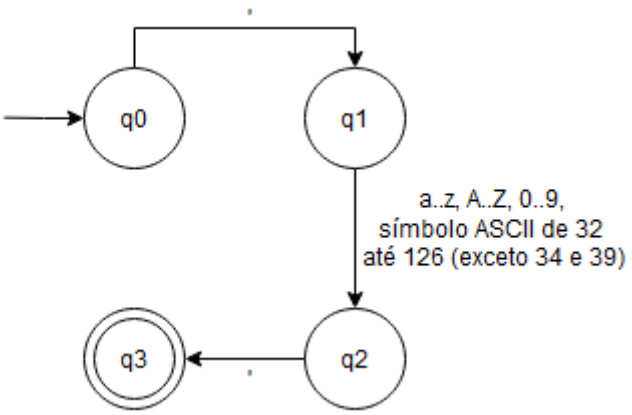

Figura 4. Diagrama de transição do autômato para *Character*

A formalização do autômato *Character* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2, q3};

Σ = {!, **'**, #, $,%,&, (,), *, +, ,, -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,v, w, x, y, z, {, | , } , ~,};

δ = {(q0, ') = q1; (q1, Σ exceto {'}) = q2; (q2, ') = q3};

s = q0;

F = {q3}.

O diagrama de transição do autômato para a identificação de *Delimiters* é exibido na Figura 5. O autômato é composto por dois estados, sendo apenas um final.

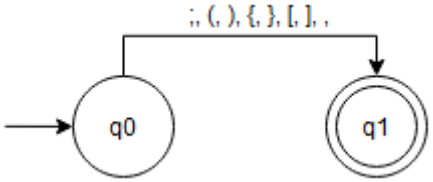

Figura 5. Diagrama de transição do autômato para *Delimiters*

A formalização do autômato *Delimiters* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1};

Σ = {;, (,), {, }, [,], ,};

δ = {(q0, Σ) = q1};

s = q0;

F = {q1}.

Os elementos classificados dentro do grupo *Comments*, isto é, comentário em bloco e comentário de linha única, foram separados em dois autômatos distintos, sendo um para comentário em bloco e outro para comentário de linha única. Na Figura 6, é exibido o diagrama de transição para o autômato identificador de comentário em bloco.

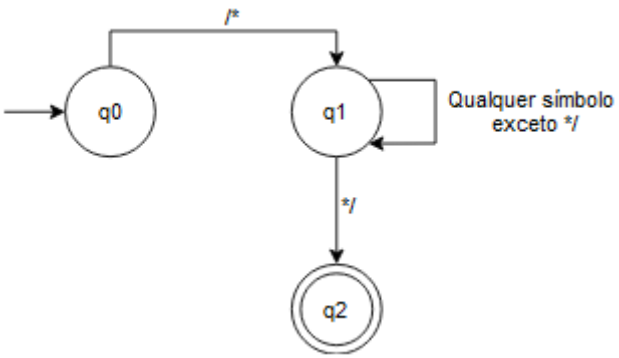

Figura 6. Diagrama de transição do autômato para comentário em bloco

A formalização do autômato comentário em bloco é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2};

Σ = {/*, */, *qualquer símbolo*};

δ = {(q0, /*) = q1; (q1, *qualquer símbolo* exceto {*/}) = q1; (q1, */) = q2};

s = q0;

F = {q2}.

No caso dos comentários, os símbolos compostos por dois elementos (/*, */ e //) foram considerados como uma única unidade na representação do autômato, de modo a

facilitar sua compreensão e construção. Além disso, para o comentário em bloco, foi utilizada a expressão "*qualquer símbolo exceto {*/}*" para indicar que, dentro do comentário, pode haver qualquer elemento, exceto o elemento de fechamento do bloco, pois sua utilização implica no término do comentário. Já para o comentário de linha única, foi utilizada a expressão "*qualquer símbolo*" para indicar que qualquer caractere é aceito dentro do comentário. Na Figura 7, é exibido o diagrama de transição do autômato identificador de comentários de linha única.

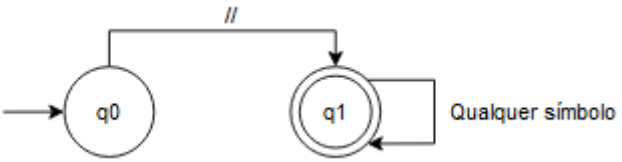

Figura 7. Diagrama de transição do autômato para comentário de linha única

A formalização do autômato comentário de linha única é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1};

Σ = {//, *qualquer símbolo*};

δ = {(q0, //) = q1; (q1, *qualquer símbolo*) = q1};

s = q0;

F = {q1}.

Para os *Operators* foram desenvolvidos um conjunto de autômatos, visto que determinados operadores são compostos por mais de um símbolo. Ao todo, foram desenvolvidos cinco autômatos distintos para a identificação de diferentes operadores. Esses autômatos foram apelidados de: *Operators 1, Operators 2, Operators 3, Operators 4* e *Operators 5.*

O autômato *Operators 1* é responsável por identificar os operadores sem combinação (* / .). Seu diagrama de transição de estados é apresentado na Figura 8.

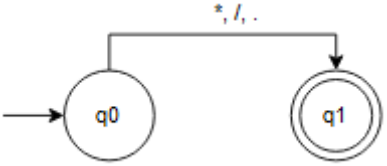

Figura 8. Diagrama de transição do autômato para *Operators 1 (Operators)*

A formalização do autômato *Operators 1* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1};

Σ = {*, /, .};

δ = {(q0, *) = q1; (q0, /) = q1; (q0, .) = q1};

s = q0;

F = {q1}.

O autômato de *Operators* 2, é responsável por identificar operadores com símbolos iguais e que são exclusivamente duplos (&& ||). O diagrama de transição de estados do autômato é apresentado na Figura 9.

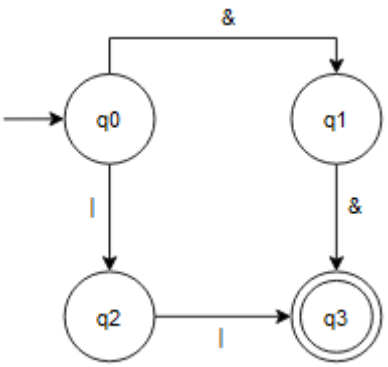

Figura 9. Diagrama de transição do autômato para *Operators 2 (Operators)*

A formalização do autômato *Operators 2* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2, q3};

Σ = {&, |};

δ = {(q0, &) = q1; (q0, |) = q2; (q1, &) = q3; (q2, |) = q3};

s = q0;

F = {q3}.

O autômato *Operators 3*, é responsável por identificar operadores de soma e subtração, que podem ser repetidos (+ - ++ --). Seu diagrama de transição de estados é apresentado na Figura 10.

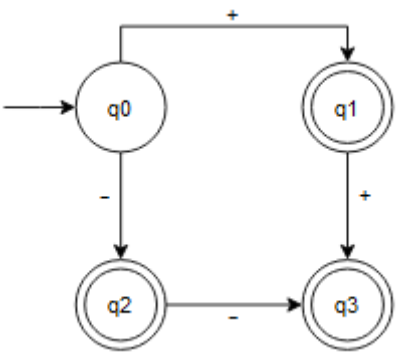

Figura 10. Diagrama de transição do autômato para *Operators 3 (Operators)*

A formalização do autômato *Operators 3* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2, q3};

Σ = {+, -};

δ = {(q0, +) = q1; (q0, -) = q2; (q1, +) = q3; (q2, -) = q3};

s = q0;

F = {q1, q2, q3}.

O autômato *Operators* 4 é responsável por identificar o operador composto do símbolo exclamação junto a uma igualdade (!=). O diagrama de transição de estados do autômato é apresentado na Figura 11.

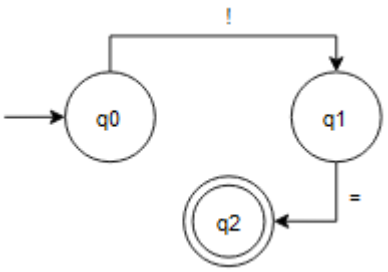

Figura 11. Diagrama de transição do autômato para *Operators 4 (Operators)*

A formalização do autômato *Operators 4* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2};

Σ = {!, =};

δ = {(q0, !) = q1; (q1, =) = q2};

s = q0;

F = {q2}.

O autômato *Operators 5*, é responsável por identificar operadores que podem vir acompanhados do sinal de igualdade (= == > >= < <=). O diagrama de transição do autômato é apresentado na Figura 12.

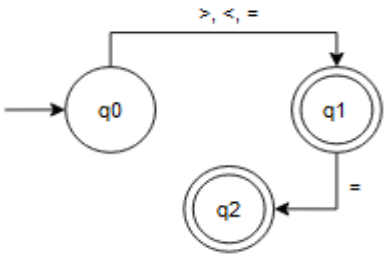

Figura 12. Diagrama de transição do autômato para *Operators 5 (Operators)*

A formalização do autômato *Operators 5* é dada por:

M = (K, Σ, δ, s, F), onde

K = {q0, q1, q2};

Σ = {>, <, =};

δ = {(q0, >) = q1; (q0, <) = q1; (q0, =) = q1; (q1, =) = q2};

s = q0;

F = {q1, q2}.

Os autômatos foram implementados em uma classe denominada *Lexical*. A classe é composta por um método principal, denominado *lex*, que é responsável por obter o símbolo atual do lexema e chamar cada um dos autômatos para a validação da palavra. Dentro dessa classe, há um conjunto de métodos que implementam os autômatos modelados. Os autômatos foram agrupados nos métodos de acordo com suas características; por exemplo, todos os autômatos responsáveis por identificar algum operador foram definidos dentro de um método comum. A classe recebe o código a ser analisado, separado em um conjunto de linhas, e retorna como saída os *tokens* encontrados, os erros e a tabela de símbolos.

Além disso, foram implementadas outras três classes: *Token*, *Erros* e *BaseTokens*. A classe *Token* é responsável por gerar um objeto do tipo *Token*. O token é o produto básico da análise léxica, onde cada lexema aceito por meio da análise é mapeado para um token. Neste caso, a classe implementada é composta por cinco atributos: *token*, *lexeme*, *position*, *line* e *ref*. O atributo *token* armazena um texto que identifica o tipo de token; *lexeme* guarda o lexema que gerou o token; o atributo *line* identifica em que linha do código o lexema foi encontrado; *position* indica a posição do lexema na linha; e o atributo *ref* é definido apenas para tokens do tipo *Identifiers*, sendo responsável por referenciar a posição do identificador na tabela de símbolos.

A classe *Erros* tem a função de gerar objetos do tipo *Erro*. Quando um erro léxico é encontrado, é criado um objeto para armazenar as informações sobre o erro, como o lexema que resultou no erro, a linha do código em que o erro foi identificado, a posição inicial do lexema na linha e uma mensagem que descreve o tipo de erro.

Já a classe *BaseTokens* produz um objeto contendo um dicionário populado com o conjunto de identificadores dos tokens básicos possíveis, associados aos seus respectivos lexemas geradores. Além disso, a classe implementa dois métodos principais: *get_token_keyword* e *get_token*. O método *get_token_keyword* verifica se determinado lexema está associado a alguma *Keyword*; se estiver, retorna o identificador referente à palavra-chave, caso contrário, retorna *None*. Já o método *get_token* é responsável por obter o identificador para todos os lexemas básicos, como operadores, delimitadores, tipos e comentários. Assim, ao encontrar um lexema válido, o analisador consulta o objeto *BaseTokens* por meio desses métodos para identificar a qual tipo de *token* o lexema é associado.

Por fim, foi desenvolvido um arquivo principal denominado *main*, responsável por exibir a interface gráfica, obter o código-fonte fornecido pelo usuário, chamar o analisador léxico e retornar o resultado da análise. Ele exibe os *tokens* gerados, os erros (se houver) e a tabela de símbolos (caso existam *tokens* de identificadores) por meio da interface gráfica.

3. Resultados, Experimentos e Análises dos Resultados

A interface gráfica desenvolvida é composta por quatro *frames* (quadros) e um único botão, de modo a conferir interatividade e simplicidade. Na Figura 13 é apresentada a interface gráfica criada, destacando cada um dos seus componentes.

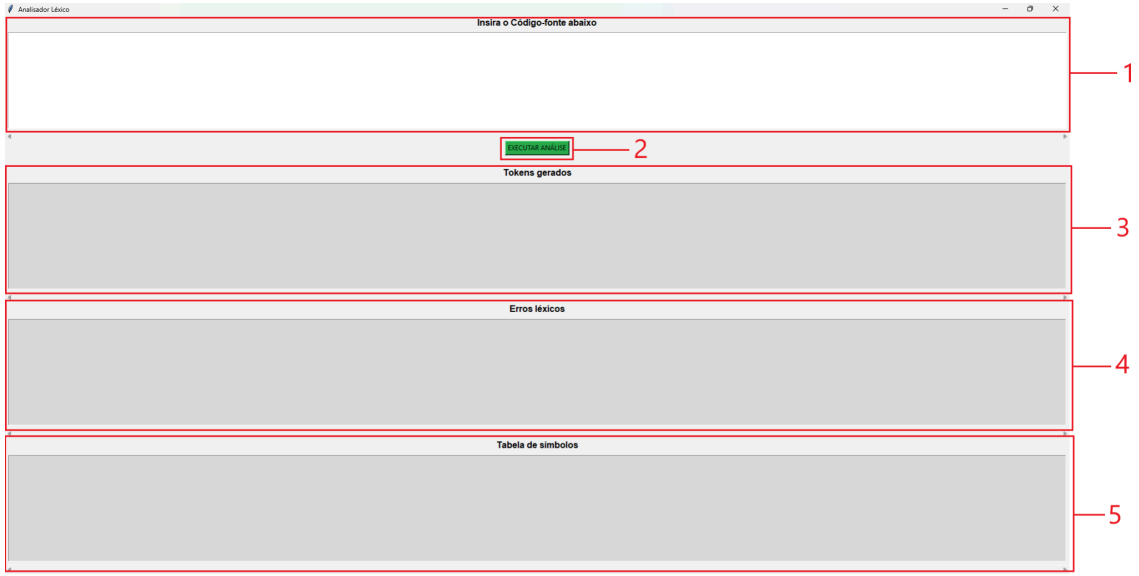

Figura 13. Componentes da interface gráfica

O primeiro item (1) da interface é uma caixa de texto, onde o código-fonte a ser submetido à análise léxica pode ser inserido. O item 2 é um botão responsável por enviar o código-fonte para a análise léxica; assim, quando o usuário terminar de inserir o código desejado, basta clicar no botão para executar a análise. O item 3 é um componente de lista, onde são exibidos os *tokens* resultantes da análise léxica. O item 4 é outro componente de lista, cuja funcionalidade é exibir os possíveis erros encontrados durante a análise. O item 5, um adicional, é um componente de lista responsável por exibir a tabela de símbolos criada durante a análise léxica. Além disso, é importante destacar que os componentes do tipo lista são incrementados de cima para baixo, apresentando um elemento por linha; assim, pode ser necessário rolar a lista (usando o botão de scroll do mouse) para visualizar todos os elementos.

Para verificar o funcionamento do analisador léxico, foram realizados diversos testes, incluindo testes específicos para cada um dos autômatos, testes gerais e testes de identificação de erros. Nos testes específicos, à medida que cada autômato era implementado, o conjunto de símbolos que ele deveria reconhecer como lexemas válidos era fornecido ao analisador. No caso das *Keywords*, foi submetido ao analisador o conjunto completo de palavras-chave. Para os *Identifiers*, diferentes variações foram testadas: algumas compostas apenas por letras, outras por letras e números, e outras por

letras, números e sublinhados (_). A mesma lógica foi aplicada para os demais autômatos. Esses testes preliminares permitiram validar o funcionamento dos autômatos implementados.

Após a validação individual dos autômatos, foram realizados testes gerais para verificar o funcionamento completo do analisador léxico. Nesse caso, submetemos ao analisador um código-fonte que continha os lexemas dos testes específicos, porém todos de uma vez, como se fossem parte de um único código-fonte. Além disso, variamos a posição de cada lexema no código. Na Figura 14, é apresentado um dos testes gerais realizados, no qual é possível observar a mensagem de sucesso, indicando que todos os elementos foram reconhecidos corretamente.

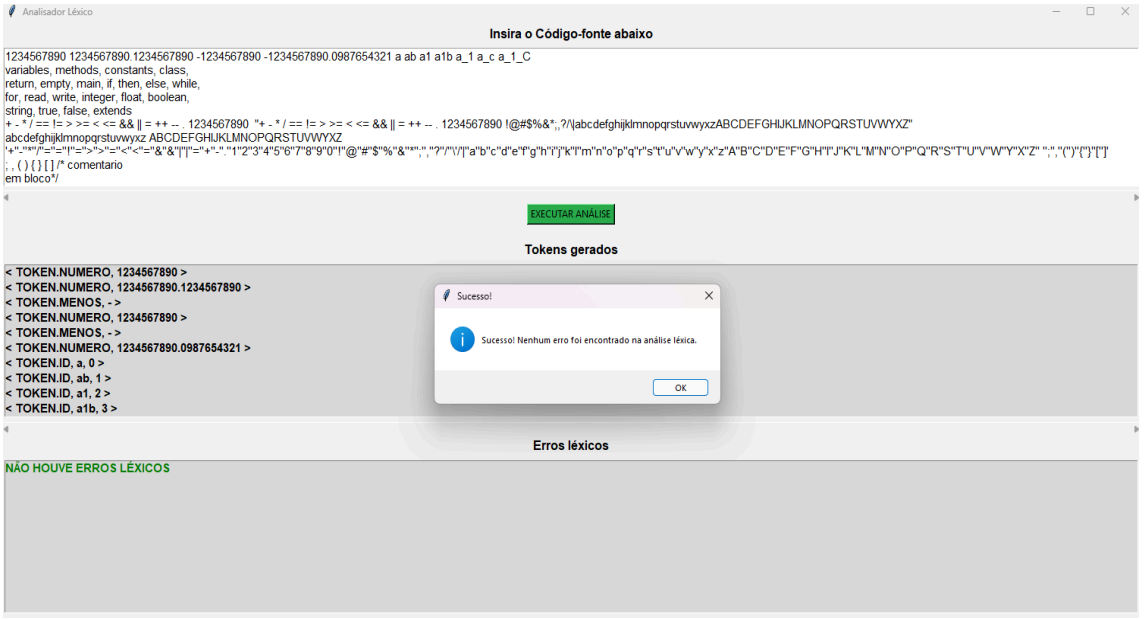

Figura 14. Exemplo de teste geral executado para validar o funcionamento do analisador léxico desenvolvido

Por fim, para comprovar o funcionamento adequado do analisador, foram submetidos ao sistema diversos códigos contendo erros léxicos, a fim de verificar se o analisador seria capaz de identificá-los. Os principais erros léxicos destacados foram: caractere desconhecido/inválido, *String* ou *Character* sem fechamento, comentário em bloco não finalizado, presença do símbolo ASCII 34 em *Character*, presença do símbolo ASCII 39 em *String*, e *Character* contendo menos ou mais de um símbolo. O sistema identificou corretamente todos os erros léxicos, como esperado. Na Figura 15, é mostrado um exemplo de teste validando a identificação dos erros. Esses testes confirmaram que o analisador foi capaz de identificar corretamente tanto os lexemas válidos quanto os erros, gerando os *tokens* correspondentes conforme o esperado.

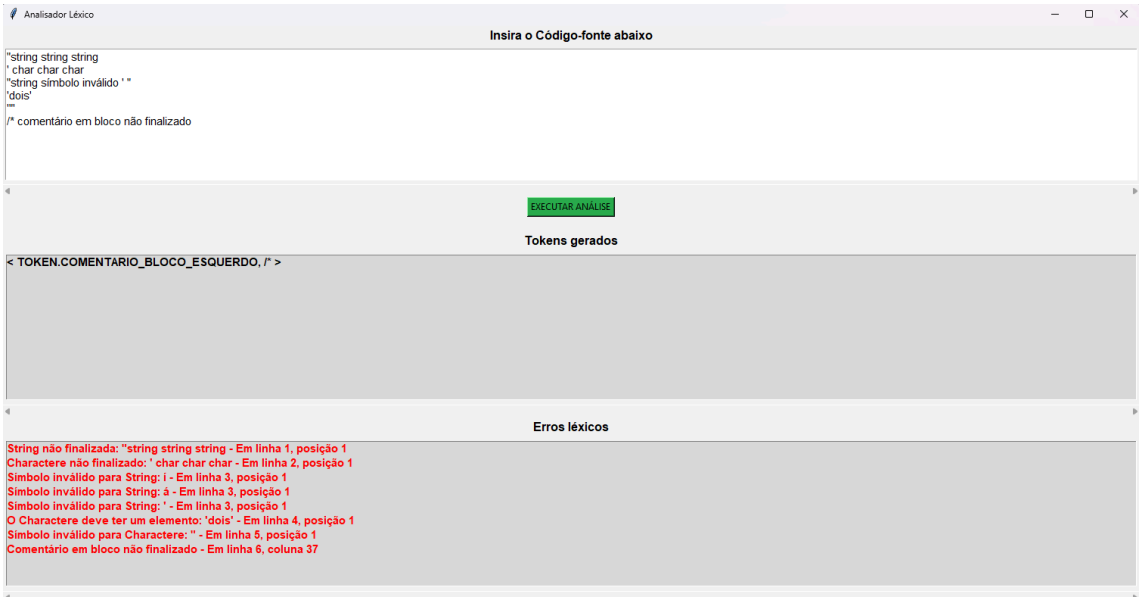

Figura 15. Exemplo de teste de identificação de erros para validar o funcionamento do analisador léxico desenvolvido

7. Conclusão

O analisador léxico desenvolvido é capaz de desempenhar as atribuições para as quais foi projetado. O sistema identifica corretamente os lexemas e gera os tokens correspondentes, além de detectar possíveis erros léxicos de forma eficiente. Além disso, a incorporação da tabela de símbolos na interface gráfica permite uma visualização clara e organizada dos identificadores.

References

Hopcroft, J. E., Motwani, R. e Ullman, J. D. (2002), Introdução à Teoria de Autômatos, Linguagens e Computação, Rio de Janeiro, Campus, 2ª edição.

Ricarte, I. L. M. (2003), Programação de Sistemas: uma introdução, São Paulo, 188 páginas.