André Luiz Belmiro de Albuquerque Junior

Compiladores

Orientador: Leonardo

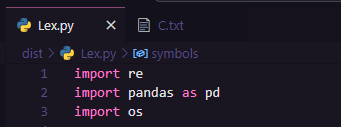
AV3 – Compilador

Conforme solicitado pelo professor Leonardo, o código criado tem como proposta a criação de um analisador Léxico e Sintatico que faça a leitura do código em C, como não foi solicitado que o código fosse criado em alguma linguagem em especifica, decidi criar o meu em Python.

1**- Analisador Léxico**

O Analisador Léxico é responsável por ler o código-fonte escrito em uma linguagem de programação e convertê-lo em uma sequência de tokens. Cada token identificado é classificado e atribuído a uma categoria específica, como operador, identificador ou número. Essa classificação é essencial, pois com ela geramos uma tabela de símbolos a qual será utilizada na próxima fase do processo de compilação, que é a análise sintática.

* 1. **Bibliotecas utilizadas**



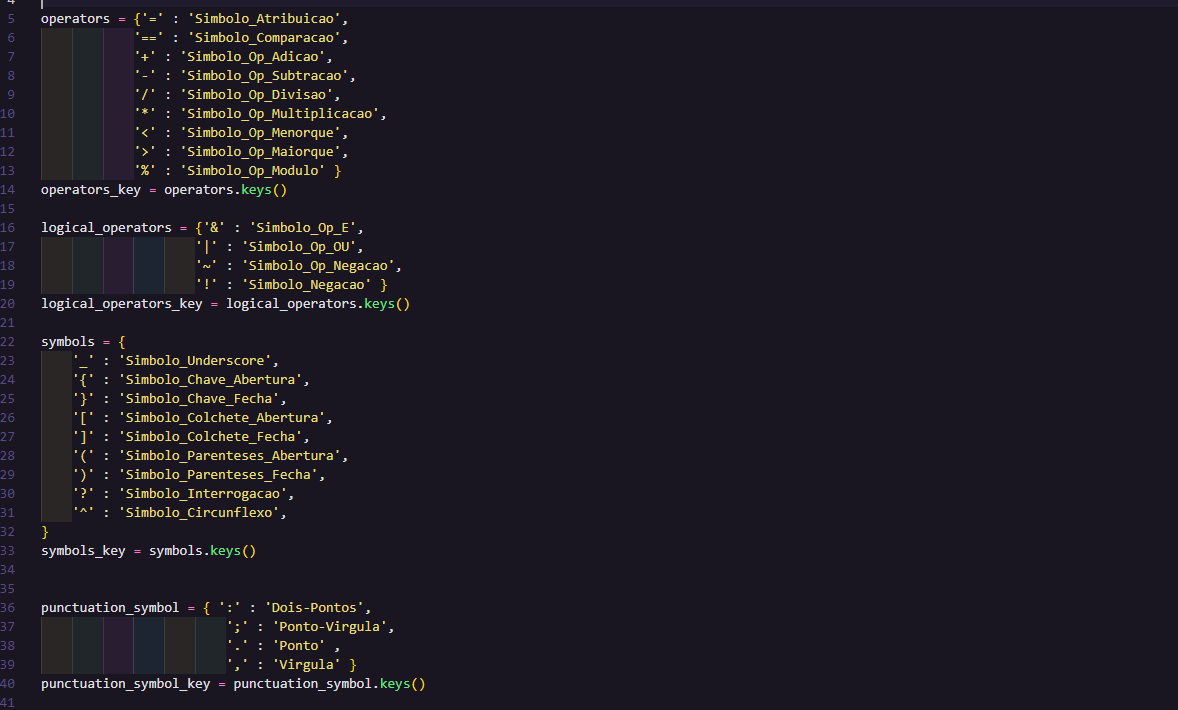
Nesse projeto foram utilizadas bibliotecas como o

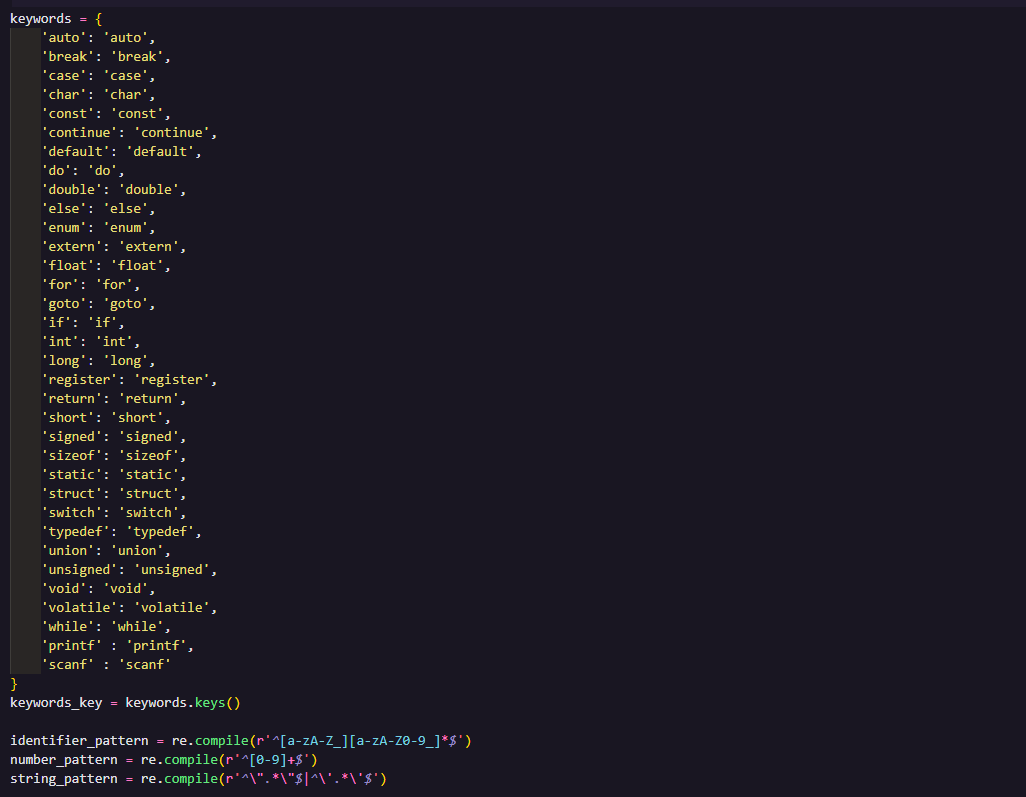
Re: um modulo do RegEx, que permitiu o uso de expressões regulares dentro do código.

Pandas: Utilizada para a manipulação e análise de dados, com ela foi possível gerar a tabela de símbolos dentro do excel.

OS: Permite a uma interação maior com o sistema operacional e suas funções, ela foi utilizada para verificar se a tabela de símbolos já foi gerada afim de evitar sobrescrever a existente.

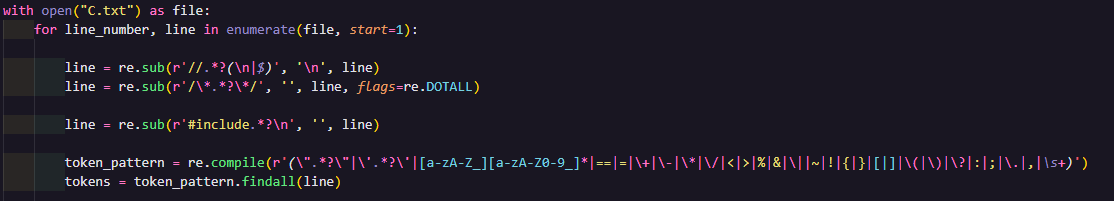
**1.2 - Tokens da linguagem C**





Aqui foi criado o dicionário da linguagem C, definido suas palavras reservadas, operadores, símbolos e etc. Além disso, foram criadas variáveis as quais armazenam as chaves dos dicionários correspondentes, essas chaves são usadas para verificar se um token corresponde a alguma estrutura do dicionário ou não.

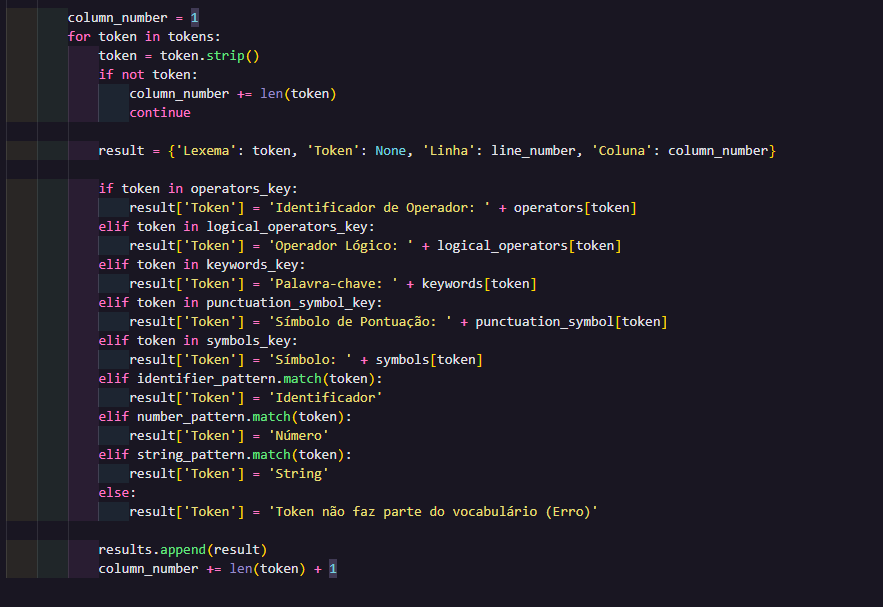
**1.3 – Abertura do código C e organização**



Nessa parte, realizamos a abertura do arquivo contendo o código C para leitura, e temos uma função **enumerate**, o qual nos permite identificar a linha original do código ao gerar a tabela de símbolos.

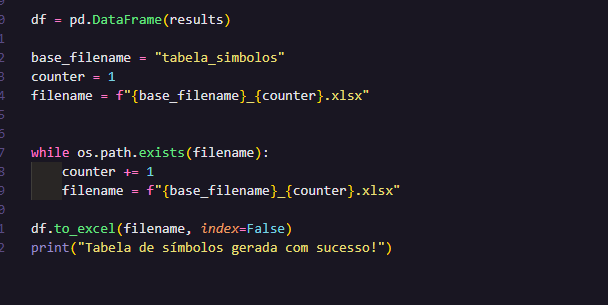
Os **line** abaixo são para a remoção de possíveis comentários ou bibliotecas que possam estar inclusas no arquivo contendo o código C. E por ultimo temos a padronização das expressões regulares para identificar diferentes tipos de tokens na linha, isso me permitiu realizar a captura de **strings**, a qual estava se mostrando uma dificuldade considerando que é possível ter espaçamento e diversos tipos de caracteres dentro de uma.

**1.4 – Leitura do código C e organização**



No início temos a utilização de um rastreamento da posição do token dentro da linha atual, junto de um loop que nos permite percorrer sobre todos os tokens obtidos da linha do código, assim removendo os espaçamentos em brancos. O **result** cria um dicionário o que será adicionado no fim a lista **results**, os dois permitirão a organização dos dados na tabela de símbolos depois. Logo em seguida temos a verificação dos elementos do código em C e sua checagem dentro do dicionário criado anteriormente.

1.5 - Transferência dos resultados para o excel.



Utilização do **panda** e do **os** para a criação do arquivo excel contendo o resultado do analisador léxico, e por fim um print sinalizando o sucesso na criação da tabela.

1. **- Código C utilizado**

Segue código C utilizado como teste, ele tem como função a soma de 2 variáveis inteiras, o código pode ser alterado dentro do arquivo txt.

#include <stdio.h>

int main() {

int num1, num2, soma;

// Solicita a entrada do primeiro número

printf("Digite o primeiro número: ");

scanf("%d", &num1);

// Solicita a entrada do segundo número

printf("Digite o segundo número: ");

scanf("%d", &num2);

// Realiza a soma dos dois números

soma = num1 + num2;

// Exibe o resultado

printf("A soma de %d e %d é: %d\n", num1, num2, soma);

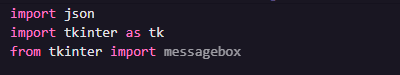
return 0;

}

1. **Analisador Sintático**

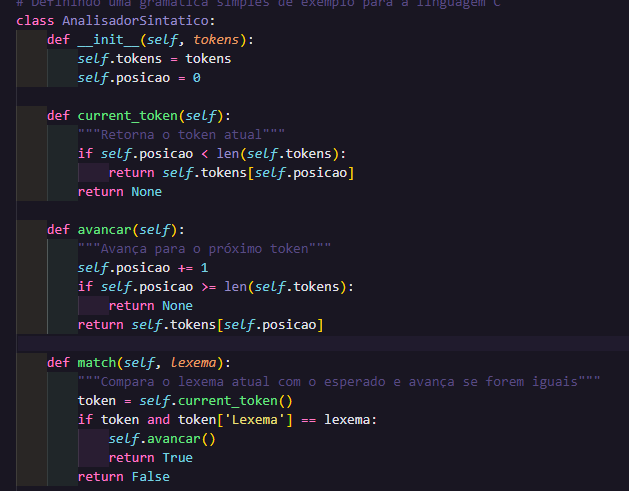
Um analisador sintático é uma etapa do processo de compilação ou interpretação de uma linguagem de programação. Ele verifica se uma sequência de tokens (produzidos por um analisador léxico) segue a gramática da linguagem. A gramática define as regras sintáticas válidas para um programa. O objetivo é identificar erros de estrutura no código e, muitas vezes, criar uma representação intermediária que será usada por etapas posteriores, como a geração de código.

* 1. **Bibliotecas utilizadas**

****

Bibliotecas como json e tkinter foram utilizadas para a leitura da tabela de símbolos a qual foi gerada tanto em formato excel, quanto em formato json (Melhor para leitura do código). O tkinter foi utilizado para a criação de uma telinha sinalizando que a analise foi um sucesso ou não.

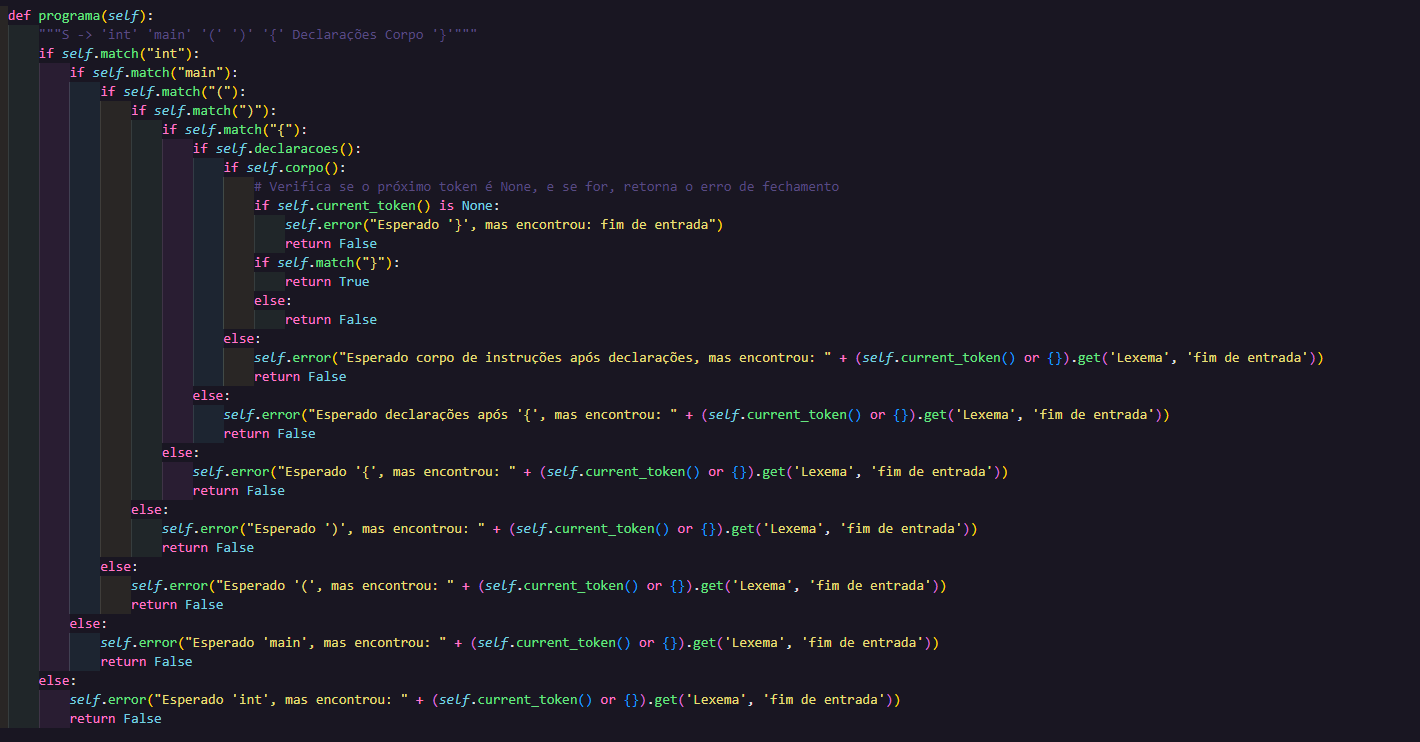
* 1. **Tratamento dos tokens**

****

Esse trecho do código serve para a realização do tratamento do token, indo desde a inicialização da classe com uma lista de tokens e define a posição inicial para análise, a retomada do token atual da análise ou None se todos os tokens já foram processados, o avanço para o próximo token na lista e retorna o novo token ou None caso tenha alcançado o final, e por último a verificação se o token atual corresponde ao lexema esperado. Se for verdadeiro, avança para o próximo token.

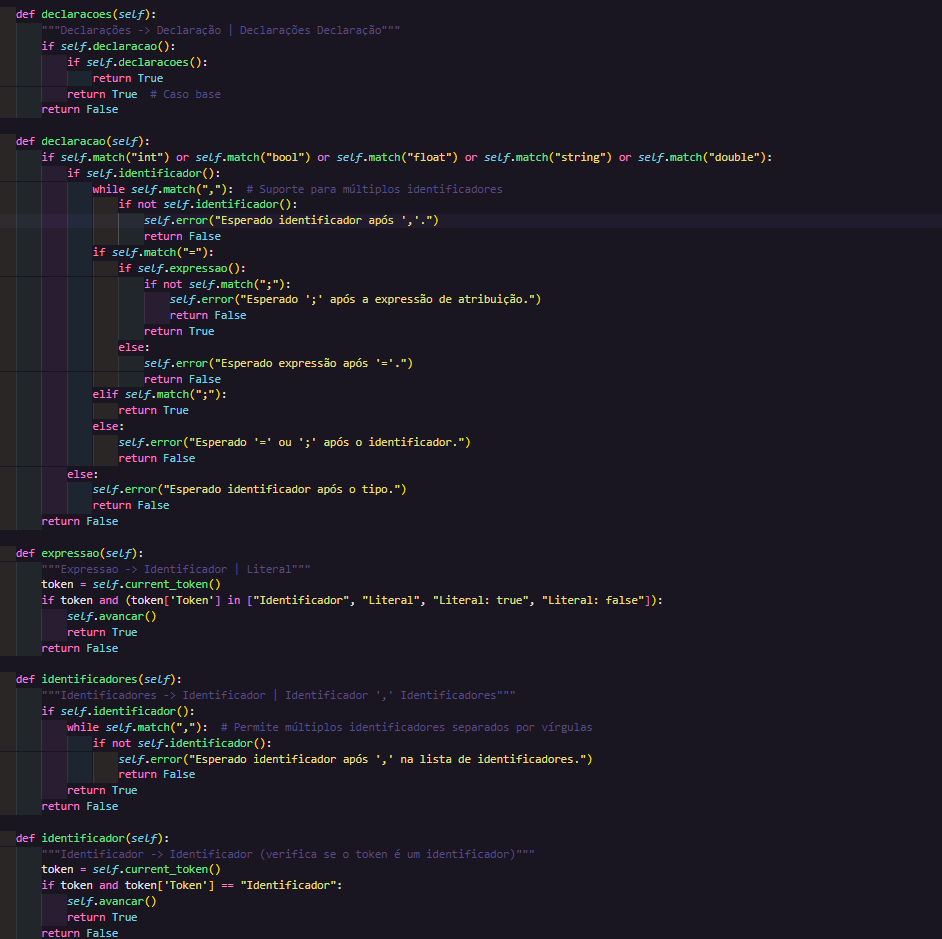
* 1. **Gramatica da linguagem C**

**3.3.1- Tratamento da Main**

****

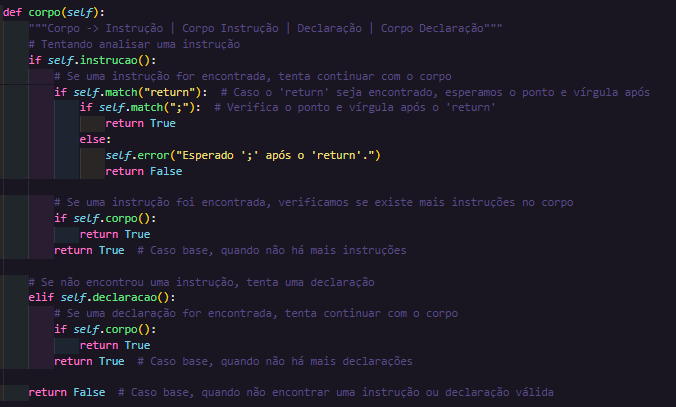
Implementa a regra inicial da gramática para um programa em C, verificando elementos da main, como int main() { ... }.

* 1. **Declarações, Identificadores e expressões**

****

Tratamento da linguagem C seguindo seus padrões para Identificadores, literais, expressões e declarações. Os tokens são verificados para verem se estão seguindo esse padrão e gerando a reportagem de erro caso não.

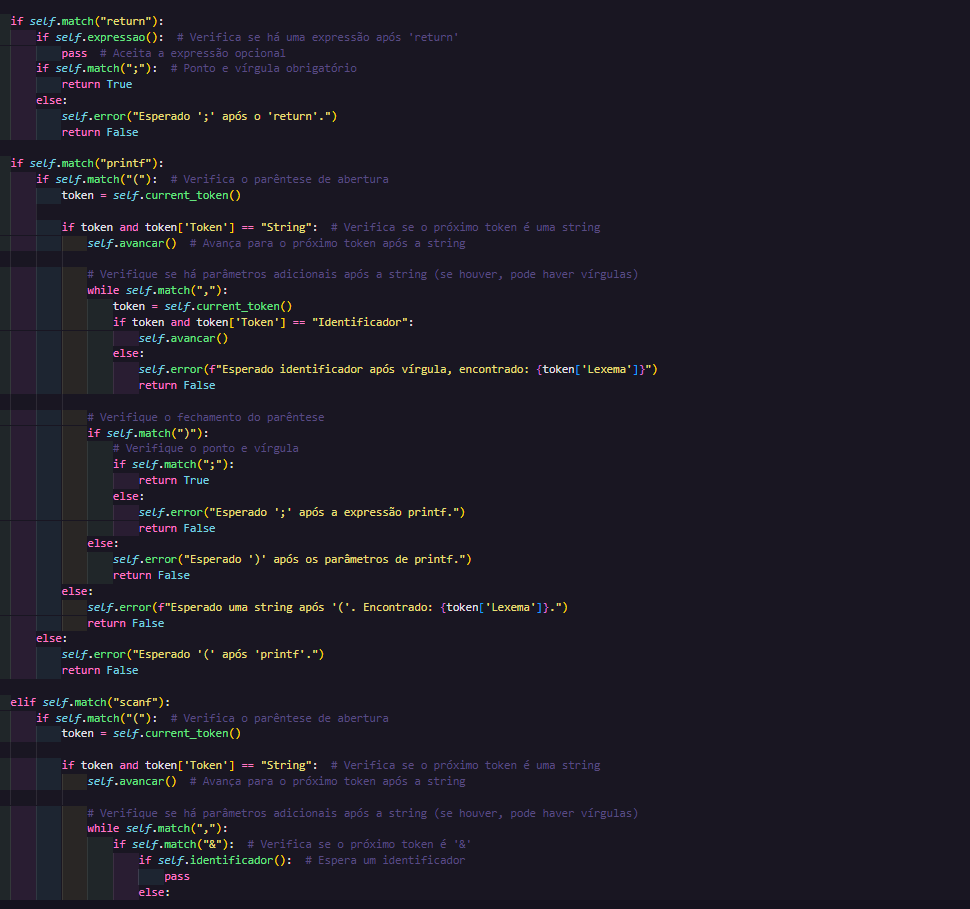
* 1. **Corpo Principal**

****

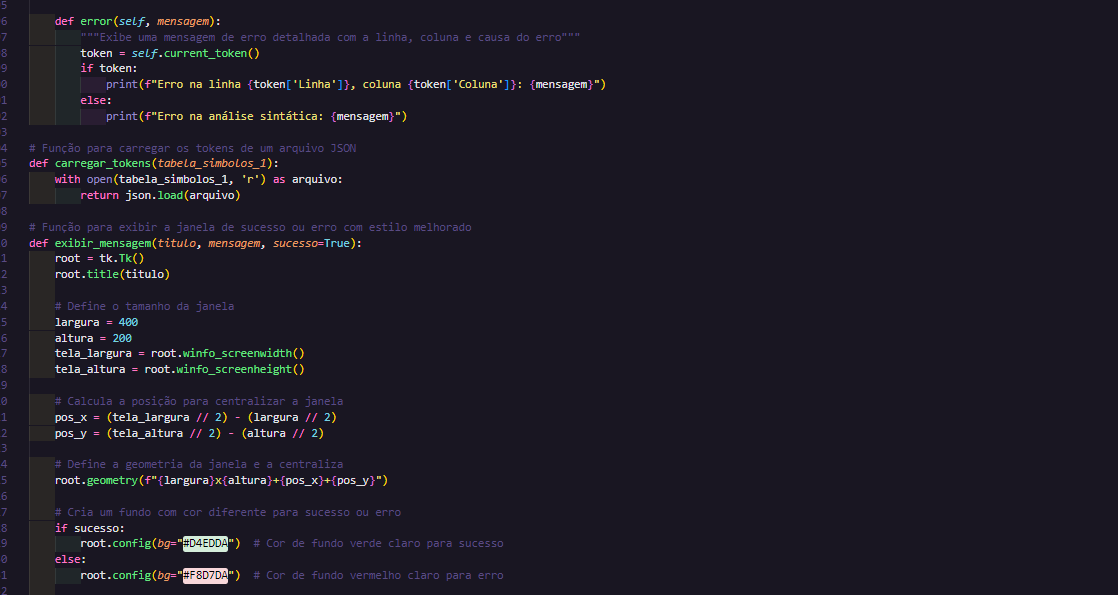
Analise do corpo principal do código.

* 1. **Instruções**

****

****

Instruções necessárias para serem seguidas em caso de uma estrutura dentro da linguagem C, dentro do código temos exemplos como printf, scanf, main, booleano, return.

* 1. **Criação da tela de mensagem, sinalização de erro, leitura arquivo json  
     **Aqui temos o tratamento de erro apontando a linha, coluna e mensagem de erro, em seguida temos a leitura da tabela de símbolos no arquivo json gerado anteriormente no léxico, código para a criação da mensagem de erro ou sucesso da análise.

1. **Tabela Símbolos**

Está é gerada e repassada pro excel para melhor visualização, segue print abaixo com exemplo.

