Projeto de Construção de Compiladores: 1º Etapa

1. Analisador Léxico:

Construir, em Java, uma classe que defina um analisador léxico (`AnLexico`). Essa classe deve possuir um método `nextToken()`, responsável por processar a entrada, caractere por caractere, e retornar um token válido. Dica: usar o FileLoader.

Ele deve:

- Ler uma cadeia de entrada de um arquivo texto identificando os seguintes tokens:

Code	Token	Padrão (informal)	Lexema
1	**NUM_INT**	literal numérico inteiro	`123,48, 3E+10`
2	**NUM_FLOAT**	literal numérico decimal	`4.8, 3.10E+10`
3	**LITERAL**	caracteres entre aspas simples	`'Carlos', 'Batata'`
4	**ID**	letra ou "_" seguida de letra, dígito ou "_"	`val, _salario, i__`
5	**REL_OP**	`$lt$|$gt$|$ge$|$le$|$eq$|$df$`	`$df$, $gt$, $le$`
6	**ADDSUB_OP**	"+" | "-"	`+, -`
7	**MULTDIV_OP**	"*" | "/"	`*, /`
8	**ATTRIB_OP**	":="	`:=`
9	**TERM**	;	`;`
10	**L_PAR**	(	`(`
11	**R_PAR**	)	`)`
12	**LOGIC_VAL**	**"true" | "false"**	`true, false`
13	**LOGIC_OP**	**"not" | "and" | "or"**	`and, not, or`
14	**TYPE**	**"bool" | "text" | "int" | "float"**	`bool, text, int`
15	**PROGRAM**	**"program"**	`program`
16	**END_PROG**	**"end_prog"**	`end_prog`
17	**BEGIN**	**"begin"**	`Begin`
18	**END**	**"end"**	`End`
19	**IF**	**"if"**	`If`
20	**THEN**	**"then"**	`Then`
21	**ELSE**	**"else"**	`Else`
22	**FOR**	**"for"**	`For`
23	**WHILE**	**"while"**	`While`
24	**DECLARE**	**"declare"**	`declare`
25	**TO**	**"to"**	`To`

Palavras reservadas

- eliminar espaços/linhas em branco, descartando-os
- eliminar comentários do tipo: `:[ comentario ]:`
- manter contagem do num. de linhas e colunas (dica: implementar no FileLoader)
- indicar linha e coluna do (início) token que gerou erro léxico
- usar a Tabela de Símbolos para diferenciar palavras reservadas e identificadores
- a cada chamada de `nextToken`, deve retornar um objeto da classe *Token:*

```
public class Token {
   private int tokenCode;  // tipo de token
   private String lexema;  // cadeia de caracteres do token
   private int lin;        // linha em que o token ocorre
   private int col;        // coluna do 1o caractere do token
   ...                     // outros atributos necessários
   ...                     // métodos
}
```

- **Nota**: reservar um *tokenCode* para representar "fim-de-arquivo" (ex. **EOF**)

2. Notas de Design:

O projeto deve ser organizado nas seguintes classes:

a. AnLexico: classe que implementa o Analisador Léxico para a linguagem estabelecida. Deve possuir o método `nextToken`, que contém a lógica de processamento do analisador léxico
b. FileLoader: classe usada pelo AnLexico, responsável pela leitura do arquivo de entrada. Possui métodos para a leitura de caracteres e controle posicional
c. Token: representa um símbolo válido para a linguagem. Seu estado deve possuir, minimamente, o código do token (estabelece o tipo de token) e o lexema extraído do arquivo de entrada
d. TabSimbolos: define a Tabela de Símbolos, usada para armazenar dados dos Identificadores (preferencialmente, um *Singleton*)
e. ErrorHandler: tratador de erros. Em geral, deve manter registro de todos os erros encontrados durante a compilação, e ser capaz de exibir um relatório formatado no final
f. AnSintatico: nesta etapa, esta classe será apenas um "boneco" para testar o AnLexico. Classe executável (método *main*) que deve ser capaz de acionar todas as partes do Analisador, desempenhando as funções abaixo:
   - receber o nome de um arquivo-fonte como argumento
   - criar instância de `AnLexico`, passando o nome desse arquivo como parâmetro
   - chamar o método `nextToken()` até que um *token* **EOF** seja produzido
   - escrever o nome de cada *token* e seu lexema (quando existir) na saída padrão, em ordem de ocorrência
   - exibir o relatório de erros, se existir algum