

Projeto de Construção de Compiladores: 1º Etapa

1. Analisador Léxico:

Construir, em Java, uma classe que defina um analisador léxico (`Lexico`). 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 duplas	"Carlos", "Batata"
4	ID	letra ou "_" seguida de letra, dígito ou "_"	val, _salario, i_
5	RELOP	<code>\$lt \$gt \$ge \$le \$eq \$df</code>	<code>\$df, \$gt, \$le</code>
6	ARIT_AS	"+" "-"	+, -
7	ARIT_MD	"*" "/"	*, /
8	ASSIGN	"<-"	<-
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: { # comentário # }
- 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 TokenType tokenType; // 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 1:** reservar um `TokenType` para representar "fim-de-arquivo" (ex. `EOF`)
- **Nota 2:** `TokenType` deve ser construído como um tipo enumerado (`enum`)

2. Notas de Design:

O projeto deve ser organizado nas seguintes classes:

- a. Lexico: 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. Sintatico: nesta etapa, esta classe será apenas um "boneco" para testar o Lexico. 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 `Lexico`, 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