# 1º TRABALHO PRÁTICO DE COMPILADORES

Professor: Alexandre Bittencourt Pigozzo
Universidade Federal de São João Del-Rei

Neste 1º trabalho, vocês devem implementar os analisadores léxico e sintático da linguagem Csmall.

O analisador léxico deverá ler o código fonte de entrada, carregar uma sequência de caracteres lidos em um buffer e fornecer um caractere de cada vez para o autômato finito determinístico do analisador.

O programa deve ser dividido nos seguintes módulos/classes:

- Um módulo IO que realiza a leitura do código fonte e carrega os caracteres lidos em um buffer.
- O módulo IO fornece uma função getNextChar() que retorna o próximo caractere lido da entrada.
- Esse módulo também deve ter um mecanismo para recarregar o buffer após ser feita a leitura de todos os caracteres do buffer.
- Um módulo ou classe AnalisadorLexico que realiza a análise léxica propriamente dita.
- O módulo AnalisadorLexico implementa o autômato finito determinístico para reconhecimento dos tokens da linguagem. Para isso, chama a função getNextChar() para obter o próximo caractere lido da entrada e realizar, com base nesse caractere, as ações definidas no autômato.
- Todos os tokens reconhecidos devem ser colocados em um vetor. Essa será a entrada para o analisador sintático.
- Cada **token** será implementado com uma classe que contém os seguintes atributos:
  - **Nome** do token (pode ser representado por uma constante inteira ou string).
  - O **lexema** que foi reconhecido para esse token (string).
  - **Linha** no código fonte de entrada (número da linha onde o lexema foi reconhecido).
- A lista de tokens da linguagem encontra-se na tabela dada abaixo. Cada token pode ser representado, por exemplo, por uma constante inteira.
- O analisador léxico deve imprimir a sequência de tokens reconhecidos.

A segunda parte do trabalho consiste em implementar um analisador sintático utilizando o método de descida recursiva.

O objetivo da fase de análise sintática é verificar se a estrutura do código fonte de entrada é uma estrutura válida de acordo com as regras sintáticas da gramática da linguagem Csmall. Além de fazer a análise sintática da cadeia de entrada, o programa deve inserir todos os identificadores declarados na Tabela de Símbolos.

A tabela de símbolos é implementada como uma tabela hash. A chave da tabela de símbolos é uma cadeia de caracteres representando um lexema e o valor é uma referência para uma struct ou para um objeto. Esse objeto que representa a entrada da tabela de símbolos possui diversos atributos dentre os quais destaca-se:

- O lexema do identificador (obtido consultando-se o objeto token do identificador em questão).
- O tipo (INT, FLOAT, etc) do identificador.
- O número da linha em que o identificador foi declarado (obtido consultando-se o objeto token do identificador em questão).
- Uma referência para o valor que será armazenado no identificador.

Ao fim da análise sintática, deverão ser impressos todos os erros sintáticos em um arquivo de erros e deverá ser impresso também o conteúdo da tabela de símbolos.

Durante a análise sintática, quando o analisador sintático encontrar algum erro, ele deve imprimir o tipo de erro e o número da linha onde ocorreu o erro. Por exemplo, caso era esperado um ID na linha 2 e o ID não foi encontrado imprima “ID esperado na linha 2”. Após encontrar um erro, o analisador sintático caminha na entrada até encontrar o token que era esperado para poder continuar a análise. Esse processo é chamado de sincronização.

O trabalho deverá ser entregue no dia **30/09/2018**. O trabalho vale **25 pontos**. O trabalho poderá ser feito em dupla. O trabalho pode ser feito em qualquer linguagem de programação, exceto em Python.

| LEXEMA | TOKEN |
|---|---|
| main | MAIN |
| int | INT |
| float | FLOAT |
| if | IF |
| else | ELSE |
| while | WHILE |
| for | FOR |
| read | READ |
| print | PRINT |
| ( | LBRACKET |
| ) | RBRACKET |
| { | LBRACE |
| } | RBRACE |
| , | COMMA |
| ; | PCOMMA |
| = | ATTR |
| < | LT |
| <= | LE |
| > | GT |
| >= | GE |
| + | PLUS |
| - | MINUS |
| * | MULT |
| / | DIV |
| [A-Za-z] ([A-Za-z] \| [0-9] \| _ )* | ID |
| [0-9] ([0-9])* | INTEGER_CONST |
| [0-9] ([0-9])*.[0-9] ([0-9])* | FLOAT_CONST |