# Compiladores, 2017/2018
# Trabalho prático, parte 1

## – Analisador Léxico –

Fernando Lobo

## 1 Introdução

O trabalho prático da disciplina consiste em modificar o compilador da linguagem Triangle, que vem descrita no livro *Programming Language Processors in Java* da autoria de David Watt e Deryck Brown, e que foi disponibilizado pelos autores. O código fonte do compilador está escrito em Java e está disponível em,

- http://www.dcs.gla.ac.uk/~daw/books/PLPJ/

O primeiro trabalho prático incide apenas sobre a análise lexical. Para tal, é suficiente ter presente a definição do léxico da linguagem, o qual vem descrito no apêndice B.8 do livro dos referidos autores. (Existe cópia do livro na biblioteca da UAlg.)

Visite o endereço web mencionado acima. Clique em 'Software' na barra do lado esquerdo e faça download do ficheiro `Triangle-tools-2.1.zip`.

Após fazer o download, faça unzip do ficheiro e deverá ficar com uma pasta chamada `tools-2.1`. Dentro dessa pasta está uma subpasta chamada `Triangle` que tem o código fonte do compilador, o qual está organizado em várias subpastas. Uma dessas subpastas chama-se `SyntacticAnalyzer` e contém a implementação do analisador sintático (analisador léxico e parser) da linguagem Triangle. A pasta tem os seguintes ficheiros:

- `Parser.java` → código do parser. Deve ser ignorado nesta 1ª parte do trabalho.
- `Scanner.java` → implementa a class `Scanner`, o analisador léxico. O método `scan()` retorna o próximo token do ficheiro de input.
- `SourceFile.java` → implementa a class `SourceFile` que interage com o ficheiro de input.
- `SourcePosition.java` → implementa a class `SourcePosition` para saber qual a localização de cada token no ficheiro.
- `SyntaxError.java` → tratamento de erros. Deve ser ignorado nesta 1ª parte do trabalho.

- `Token.java` → implementa a class `Token`. É aqui que estão definidos os vários tipos de token da linguagem.

Para lhe facilitar a vida, é-lhe fornecido um ficheiro chamado `TestScanner.java`, feito por mim, que se limita a chamar o analisador léxico. O ficheiro `tp1-code.zip` disponível na página da tutoria eletrónica da disciplina, contém os ficheiros relevantes para a realização do 1º trabalho prático: `TestScanner.java`, `Scanner.java`, `SourceFile.java`, `SourcePosition.java`, `Token.java`. O ficheiro zip contém também um exemplo de input, `prog1.tri`, que pode ser usado para testar o analisador léxico. Obviamente que pode e deve testar com outros exemplos de input.

Antes de realizar o trabalho, teste o analisador léxico que foi posto à sua disposição. Para tal, deverá compilar os ficheiros java e executar o programa `TestScanner`. Para compilar,

```
javac *.java
```

Para testar o analisador léxico com o ficheiro de input `prog1.tri` deverá escrever na linha de comando,

```
java TestScanner prog1.tri
```

A título de exemplo, se o conteúdo de `prog1.tri` for o seguinte,

```
! This program is useless
! except for illustration.
let
   var n: Integer;
   var c: Char
in
   begin
      c := '&';
      n := n+1
   end
```

o output será,

```
Kind=let, spelling=let, position=(3, 4)
Kind=var, spelling=var, position=(4, 4)
Kind=<identifier>, spelling=n, position=(4, 4)
Kind=:, spelling=:, position=(4, 4)
Kind=<identifier>, spelling=Integer, position=(4, 4)
Kind=;, spelling=;, position=(4, 5)
Kind=var, spelling=var, position=(5, 5)
Kind=<identifier>, spelling=c, position=(5, 5)
Kind=:, spelling=:, position=(5, 5)
Kind=<identifier>, spelling=Char, position=(5, 6)
Kind=in, spelling=in, position=(6, 7)
```

```
Kind=begin, spelling=begin, position=(7, 8)
Kind=<identifier>, spelling=c, position=(8, 8)
Kind=:=, spelling=:=, position=(8, 8)
Kind=<char>, spelling='&', position=(8, 8)
Kind=;, spelling=;, position=(8, 9)
Kind=<identifier>, spelling=n, position=(9, 9)
Kind=:=, spelling=:=, position=(9, 9)
Kind=<identifier>, spelling=n, position=(9, 9)
Kind=<operator>, spelling=+, position=(9, 9)
Kind=<int>, spelling=1, position=(9, 10)
Kind=end, spelling=end, position=(10, 10)
Kind=, spelling=, position=(10, 10)
```

O output tem uma linha por cada token, com informação sobre o seu tipo (kind), valor (spelling), e posição no ficheiro de input.

# 2 Tarefas a realizar

1. Altere o código fonte de modo a que a linguagem passe também a ter como tokens mais 5 palavras reservadas: `for`, `from`, `to`, `downto`, `case`.

2. No ficheiro `Token.java` altere a string correspondente a `EOT` no array de strings `tokenTable`. A string vazia deverá ser substituida pela string "`<eot>`".

# 3 Prazo e entrega do trabalho

**Prazo:** 15/Mar/2018

**Como entregar:** A submissão deve ser feita através do Mooshak, concurso Comp1718, problema B.

O ficheiro a submeter deve ter forçosamente chamar-se `src.zip`. Note ainda que ao descompactar esse ficheiro deve ser criada uma pasta chamada `src` contendo os ficheiros `.java`. Se não fizer isso o mooshak dará erro.

**Cotação máxima:** 10 pontos (vale muito pouco porque a resolução é trivial.)

O código fornecido em `tp1-code.zip` lê o input de um ficheiro de texto. O mooshak está apenas preparado para aceitar input do Standard Input. O ficheiro `tp1-code-mooshak.zip` está preparado para ler o input do Standard Input em vez de ler de um ficheiro. Para efeitos de teste nos vossos computadores, podem fazer o seguinte na linha de comando.

```
java TestScanner < prog1.tri
```

O sinal < serve para redireccionar o input. O programa vai ler o conteúdo de `prog1.tri`, tal e qual como se o utilizador estivesse a teclar o seu conteúdo manualmente.

# 4 Exemplo de input e output no Mooshak

## Input

```
! This program is useless
! except for illustration.
let
   var n: Integer;
   var c: Char
in
   begin
      c := '&';
      n := n+1;
      for i from 2 to n do
         putint(i)
   end
```

## Output

```
Kind=let, spelling=let, position=(3, 4)
Kind=var, spelling=var, position=(4, 4)
Kind=<identifier>, spelling=n, position=(4, 4)
Kind=:, spelling=:, position=(4, 4)
Kind=<identifier>, spelling=Integer, position=(4, 4)
Kind=;, spelling=;, position=(4, 5)
Kind=var, spelling=var, position=(5, 5)
Kind=<identifier>, spelling=c, position=(5, 5)
Kind=:, spelling=:, position=(5, 5)
Kind=<identifier>, spelling=Char, position=(5, 6)
Kind=in, spelling=in, position=(6, 7)
Kind=begin, spelling=begin, position=(7, 8)
Kind=<identifier>, spelling=c, position=(8, 8)
Kind=:=, spelling=:=, position=(8, 8)
Kind=<char>, spelling='&', position=(8, 8)
Kind=;, spelling=;, position=(8, 9)
Kind=<identifier>, spelling=n, position=(9, 9)
Kind=:=, spelling=:=, position=(9, 9)
Kind=<identifier>, spelling=n, position=(9, 9)
Kind=<operator>, spelling=+, position=(9, 9)
Kind=<int>, spelling=1, position=(9, 9)
Kind=;, spelling=;, position=(9, 10)
Kind=for, spelling=for, position=(10, 10)
Kind=<identifier>, spelling=i, position=(10, 10)
Kind=from, spelling=from, position=(10, 10)
Kind=<int>, spelling=2, position=(10, 10)
Kind=to, spelling=to, position=(10, 10)
```

```
Kind=<identifier>, spelling=n, position=(10, 10)
Kind=do, spelling=do, position=(10, 11)
Kind=<identifier>, spelling=putint, position=(11, 11)
Kind=(, spelling=(, position=(11, 11)
Kind=<identifier>, spelling=i, position=(11, 11)
Kind=), spelling=), position=(11, 12)
Kind=end, spelling=end, position=(12, 12)
Kind=<eot>, spelling=, position=(12, 12)
```