Bloco 2

Introdução ao ANTLR

Resumo:

- Instalação e utilização do ANTLR.
- Construção de gramáticas simples.
- Introdução à programação em ANTLR.

Exercício 2.1
Comece por instalar o ANTLR (`http://www.antlr.org`).

Defina a seguinte gramática no ficheiro `Hello.g4`:

```
grammar Hello;                  // Define a grammar called Hello
greetings : 'hello' ID ;        // match keyword hello followed by an identifier
ID : [a-z]+ ;                   // match lower-case identifiers
WS : [ \t\r\n]+ -> skip ;       // skip spaces, tabs, newlines, \r (Windows)
```

a) Experimente compilar a gramática `Hello.g4`, e executar o tradutor resultante.

b) Altere a gramática de modo a incluir uma acção no fim da regra `greetings` escrevendo em português: `Olá <ID>`

c) Acrescente uma regra de despedida à gramática (`bye ID`), fazendo com que a gramática aceite uma qualquer das regras (`greetings`, ou `bye`). (Para definir em `ANTLR` uma regra com mais do que uma alternativa utiliza-se o separador `|`, por exemplo: `r: a | b ;`.)

d) Generalize os identificadores por forma a incluir letras maiúsculas e a permitir nomes com mais do que um identificador. (Em `ANTLR` uma regra com uma ou mais repetições e definida por `r+`, por exemplo: `r: a+ ;`.)

e) Generalize a gramática por forma a permitir a repetição até ao fim do ficheiro das regras de qualquer uma das regras atrás descritas (`greetings`, ou `bye`).

Exercício 2.2
Considere a seguinte gramática (`Calculator.g4`):

```
grammar Calculator;

program:
      stat* EOF
    ;

stat:
      expr NEWLINE
    | NEWLINE
    ;

expr:
      expr op=('*'|'/') expr
    | expr op=('+'|'-') expr
    | INT
    | '(' expr ')'
    ;

INT: [0-9]+;
NEWLINE: '\r'? '\n';
WS: [ \t]+ -> skip;
```

a) Experimente compilar esta gramática e executar o tradutor resultante.

b) Utilizando esta gramática e acrescentando acções e atributos nas regras, tente implementar uma calculadora para as operações aritméticas elementares definidas (ou seja, que efetue os cálculos e apresente os resultados para cada linha processada).

Exercício 2.3
Implemente uma gramática para fazer a análise sintáctica dos ficheiros utilizados no exercício 1.3. Utilizando um *listener*, altere a resolução desse problema por forma a incluir esta gramática na solução.

Exercício 2.4
Altere o problema 2.2 acrescentando a possibilidade de definir e utilizar variáveis. Para esse fim considere uma nova instrução (i.e. um `stat`):

```
assignment: ID '=' expr NEWLINE;
...
ID: [a-zA-Z_]+ ;
```

Para dar suporte ao registo dos valores associados a variáveis, utilize um *array* associativo (java.util.HashMap). Note também que na definição de gramáticas em `ANTLR` pode-se indicar código a ser colocado no preâmbulo do código gerado, e novos membros a serem adicionados às classes geradas:

```
@header { ··· }
@members { ··· }
```

Exercício 2.5
Descarregue a gramática da linguagem `Java` (`https://github.com/antlr/grammars-v4`), e experimente fazer a análise sintáctica de programas `Java` simples (versão 8).

Utilize o suporte para *listeners* do `ANTLR` para escrever o nome da classe e dos métodos sujeitos a análise sintáctica.

Exercício 2.6
Utilizando a gramática definida no problema 2.4 e recorrendo aos *visitors* do `ANTLR`, converta uma expressão aritmética infixa (operador no meio dos operandos), numa expressão equivalente sufixa (operador no fim). Por exemplo:

- $2 + 3 \quad \rightarrow \quad 2\ 3\ +$
- $2 + 3 * 4 \quad \rightarrow \quad 2\ 3\ 4\ *\ +$
- $3 * (2 + 1) + (2 - 1) \quad \rightarrow \quad 3\ 2\ 1\ +\ *\ 2\ 1\ -\ +$

Note que em `ANTLR` podemos associar diferentes *callbacks* a diferentes alternativas numa regra sintáctica:

```
r : a #altA
  | b #altB
  | c #altC
  ;
```

Neste caso, irão ser criados *callbacks* quer nos *listeners* quer nos *visitors*, para as três alternativas apresentadas, não aparecendo o *callback* para a regra `r`.

Exercício 2.7
Pretende-se desenvolver uma calculadora simples para operações sobre conjuntos. Nesta versão simplificada, vamos restringir os conjuntos a listas finitas de elementos, definidos por extensão.

Desenvolva uma gramática para esta calculadora, tendo em consideração a seguinte especificação:

- Um conjunto é definido por uma sequencia de palavras, ou de números, separada por vírgulas e delimitada por chavetas:

  $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, $\{1, 3, 5, 7, 9\}$

- Uma palavra é uma sequência de letras minúsculas.
- Um número é uma sequência de dígitos, eventualmente precedida pelo sinal menos ou pelo sinal mais.

- Pode-se definir (ou redefinir) variáveis que representem conjuntos com uma instrução de atribuição de valor:

  $\mathrm{C} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$

- Uma variável é uma sequência de letras maiúsculas.

- Implemente as operações (com prioridade crescente) sobre conjuntos: união (definida pelo símbolo +), intercepção (símbolo &) e diferença (símbolo \).

  $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\} \ + \ \{\mathrm{b}, \mathrm{d}\}$

- Para poder definir diferentes precedências, implemente os parêntesis.

  $(\{\mathrm{a}\} \ + \ \{\mathrm{b}\}) \setminus \{\mathrm{b}\}$

- Implemente comentários de linha definidos pelo prefixo --.

  ```
  -- this is a comment!
  ```

- Considere que a calculadora funciona como interpretador, em que cada linha representa uma instrução cujo resultado será um conjunto que deve ser apresentado:

  $\mathrm{C} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$

  result: $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$

  $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\} \ + \ \{\mathrm{b}, \mathrm{d}\}$

  result: $\{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$

  $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\} \ \& \ \{\mathrm{b}, \mathrm{d}\}$

  result: $\{\mathrm{b}\}$

  $\{\mathrm{a}, \mathrm{b}, \mathrm{c}\} \setminus \{\mathrm{b}, \mathrm{d}\}$

  result: $\{\mathrm{a}, \mathrm{c}\}$

  $\mathrm{C} \setminus \mathrm{C}$

  result: $\{\}$