



Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
Linguagens Formais e Autómatos / Compiladores
(ano letivo de 2020-2021)

Resolução parcial guiada do exercício 2.1

abril de 2021

Considera-se que as ferramentas ANTLR foram devidamente instaladas e que o leitor conhece minimamente os seus papeis.

Sugere-se que cada exercício seja resolvido numa pasta diferente. Nesse sentido, a resolução aqui apresentada, correspondendo à do exercício 1 do bloco 2, é feita numa pasta chamada **b2_1**. O formato usado para alguns comandos ANTLR pressupõe esta estruturação.

Exercício 2.1

Criação, compilação e teste do primeiro programa em ANTLR.

(a) Edição da gramática

- Na pasta **b2_1**, use o editor da sua preferência, para criar um ficheiro chamado **Hello.g4** com o conteúdo seguinte:

```
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)
```

O nome do ficheiro deve coincidir com o nome da gramática (**Hello** neste caso). Não precisa de copiar os comentários. Apenas foram colocados para clarificação. Não faça *copy&paste*. Por um lado, ao escrever está a memorizar. Por outro lado, ao copiar e colar pode inserir caracteres não ASCII.

(b) Compilação da gramática

- Abra um terminal na pasta **b2_1**.
- Execute o comando

```
antlr4 Hello.g4
```

- Execute o comando

```
ls -l
```

e comprove que foram gerados vários ficheiros Java.

- Compile todos esses ficheiros Java

```
javac *.java
```

(c) Teste usando o comando **antlr4-test**

- Execute o comando

```
antlr4-test
```

inserindo de seguida a expressão **hello mundo** seguida de ENTER e ^D (control D). Nada aconteceu? Ótimo, pois significa que a entrada inserida está de acordo com a gramática.

- Repita o comando anterior, inserindo de seguida a expressão **hello Mundo** seguida de ENTER e ^D. Algo aconteceu agora. Provavelmente apareceu a mensagem

```
line 1:6 token recognition error at: M
```

indicando que há um erro sintático na expressão inserida, na linha 1, coluna 6. O ID está definido como uma sequência de letras minúsculas e **Mundo** não encaixa nesse padrão.

- Repita novamente o comando anterior, inserindo de seguida a expressão **hola mundo** seguida de ENTER e ^D. Algo voltou a acontecer, mas diferente. Provavelmente apareceu a mensagem

```
line 1:0 missing hello at hola
```

indicando que há um erro sintático na expressão inserida, embora não o mesmo que anteriormente, na linha 1, coluna 0. A gramática obriga a que a entrada comece por **hello**, o que não aconteceu.

- Pode fazer uso do *piping* de comandos em **bash** para invocar o programa e passar-lhe a entrada na mesma linha. Comprove isso, executando o comando

```
echo "hello mundo" | antlr4-test
```

- Execute o comando

```
echo "hello mundo" | antlr4-test -tokens
```

Aparece a sequência de *tokens* correspondente à entrada inserida.

- Execute o comando

```
echo "hello mundo" | antlr4-test -gui
```

Aparece uma representação gráfica da árvore sintática correspondente à entrada inserida. Compare-a com a gramática.

(d) Criação de um programa principal em Java, usando um *visitor*

- Tipicamente é necessário criar um programa em Java associado à gramática em ANTLR. Há formas diferentes de o fazer. Neste exercício vamos fazê-lo usando um *visitor*. Execute o comando

```
antlr4-visitor Execute String
```

e comprove que foi criado o ficheiro **Execute.java**. Este ficheiro representa uma classe em Java que pode ser usada para acrescentar ações à gramática. Mais à frente iremos editá-lo para o fazer.

- Execute agora o comando

```
antlr4-main -v Execute
```

e comprove que foi criado o ficheiro **HelloMain.java**. Este ficheiro é um programa em Java associado à gramática **Hello**, que invoca o *visitor* **Execute.java**.

(e) Definição de ações

- Edite o ficheiro **Execute.java**, usando o seu editor preferido, e altere o método **visitGreetings**, acrescentando-lhe a linha

```
System.out.println("Olá " + ctx.ID());
```

O método **visitGreetings** está associado à única produção **greetings** da gramática. O parâmetro **ctx** representa o contexto associado a essa produção. A expressão **ctx.ID()** representa aquilo que for ‘apanhado’ pelo *token* ID. Deve ter visto atrás, quando executou o comando **antlr4-test -gui**, que **greetings** corresponde à raiz da árvore sintática. O método **visitGreetings** é por isso invocado automaticamente pelo programa **main** gerado anteriormente.

(f) Compilação e execução

- Uma vez que se optou pela utilização de *visitors*, é preciso indicar essa opção na compilação da gramática. Para isso, execute o comando

```
antlr4 -visitor Hello.g4
```

- De seguida, compile todos os programas Java

```
javac *.java
```

- Como alternativa à execução dos dois últimos comandos, basta executar o comando

```
antlr4-build
```

- Finalmente, execute o programa, dando-lhe a entrada **hello mundo**, e veja o resultado.

```
echo "hello mundo" | antlr4-java HelloMain
```

- Em alternativa, pode executar

```
echo "hello mundo" | antlr4-run
```

(g) Generalização

- A generalização aqui abordada não implica qualquer alteração à gramática. O que se faz é uma alteração do programa principal de modo a invocar o *parser* da gramática para cada linha de entrada.

- Execute o comando

```
antlr4-main -v Execute -i -f
```

A opção `-i` instroí o comando para criar uma versão interativa do programa principal. Por defeito, o comando `antlr4-main` falha se já existir um ficheiro `main`. A opção `-f` força a criação de um novo.

- Volte a compilar, execute e veja o resultado.

```
antlr4-build  
antlr4-run
```

Aqui, optou-se pela invocação direta (sem o *piping*), porque a entrada pode ser multi-linha, havendo ação por cada linha. Para sair faça `^D`.