# Princípios de Programação

# Trabalho Individual 2

Licenciatura em Engenharia Informática

2016/2017

Neste trabalho devem escrever um programa Haskell que expande listas através de informação especificada por uma lista de **regras**. Uma regra é um par (elemento, lista de elementos). A primeira componente da regra designa-se por **chave**, não havendo duas chaves iguais. A segunda componente da regra designa-se por **expansão** da respectiva chave. Os elementos que compõem regras podem ser de qualquer tipo desde que sejam equiparáveis. Como representação das regras vamos incluir no código um sinónimo de tipo:

```
type Rules a = [(a,[a])]
```

Como funciona o processo de expansão? A partir de uma lista base designada **axioma**, o programa verifica, um a um, cada elemento do axioma. Para cada elemento do axioma que seja uma chave, i.e., tenha uma regra associada, substitui-se esse elemento pela sequência de elementos da respectiva expansão. Um elemento que não tenha expansão permanece inalterado. Quando fizermos todas as expansões dos elementos do axioma, terminamos uma **iteração**. O processo pode então recomeçar numa nova iteração onde o resultado da iteração anterior passa a ser o novo axioma.

A função `turtle` será a responsável por este processo. Um exemplo de uso com inteiros:

```
ghci> regras = [(1,[2,1]), (2,[0,2,1])]
ghci> axioma = [1]
ghci> turtle regras axioma 1
[2,1]
ghci> turtle regras axioma 2
[0,2,1,2,1]
ghci> turtle regras axioma 3
[0,0,2,1,2,1,0,2,1,2,1]
ghci> turtle regras axioma 0
[1]
```

Outro exemplo agora com *chars*:

```
ghci> turtle [('X',"X+YF+"),('Y',"-FX-Y")] "FX" 2
"FX+YF++-FX-YF+"
```

Como se pode concluir destes exemplos, a assinatura de `turtle` deverá ser `turtle :: Eq a => Rules a -> [a] -> Int -> [a]`.

Para além da implementação de `turtle` pede-se igualmente a função auxiliar `getExpansion` – que será útil para definir a função `turtle` – com assinatura `getExpansion :: Eq a => Rules a -> a -> [a]` que, dado uma lista de regras e um elemento, devolve a respectiva expansão, ou devolve o próprio elemento numa lista se não existir expansão associada. Por exemplo:

```
ghci> getExpansion regras 1
[2,1]
ghci> getExpansion regras 0
[0]
```

A função `getExpansion` deve ser implementada através de uma função *fold*. A implementação de `turtle` deve igualmente usar funções de ordem superior, nomeadamente a função `map` e outras que achar conveniente. Podem criar mais funções auxiliares se necessário.

**Nota final.** Este processo pode ser usado para produzir desenhos geométricos num sistema de programação Logo. O Logo é uma ferramenta pedagógica de ensino à programação que usa uma tartaruga (daí o nome da nossa função!) para fazer desenhos geométricos baseado em comandos muito simples. Podem usar esta página para observar um sistema Logo em funcionamento[1]. Nesta página podem calcular iterações de axiomas e regras e, assim, testar a correção das vossas soluções.

**Notas**

1. Deve juntar a assinatura para cada função que escrever.
2. Pode usar as funções do **`Prelude`**.
3. Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 14 de Novembro de 2016. O nome do ficheiro deve ter o formato `t2_fcXXXXX.hs`, sendo `XXXXX` o seu número de aluno.
4. Os trabalhos serão avaliados semi-automaticamente. Respeite os nomes e assinaturas das funções.

[1] O link é `http://www.di.ciencias.ulisboa.pt/~jpn/pp/turtle.html`.