CI062 : Técnicas alternativas de programação **Prova 2**
2º semestre 2017

Instruções para a prova

- A prova é sem consulta;
- A prova dura 1 hora e 30 minutos;
- A prova deve ser implementada usando unicamente programação funcional.

Questão 1 : (20 pontos)
Explique porque não há estado compartilhado em programas escritos usando o paradigma funcional.

Questão 2 : (40 pontos)
Considere a função *similar*, que recebe como entrada três parâmetros: uma lista L1, uma lista L2 e uma função de similaridade F:(Any=>Boolean). A avaliação desta função *similar* retorna verdadeiro se, os elementos da lista L1 forem similares aos elementos da lista L2, 1 a 1, na mesma ordem. As duas listas tem o mesmo tamanho, isto não precisará ser testado. Veja as observações abaixo.
a) Implemente uma função *similar* usando recursão.
b) Qual o tipo de recursão usada? Explique o porquê desta escolha.
c) Dê um exemplo de chamada da função criada, para uma lista L1 e L2 com 4 elementos cada.

Observações:
- não é permitido usar as funções de percurso de lista nativas de linguagens de programação (ex, *reduceLeft, reduceRight, filter, forAll, exists, forEach*)
- as seguintes funções de acesso a elementos de uma lista podem ser usados: *tail(l : List)* – retorna a lista, menos o primeiro elemento; *head(l : List)* – retorna o primeiro elemento da lista; *isEmpty(l : List)* – retorna "true " se a lista está vazia; *size(l : List)* – retorna o tamanho da lista; *last(l : List)* – retorna o último elemento

Questão 3 : (20 pontos)
Crie uma chamada de função que receba uma lista de Strings L como parâmetro e que retorna o maior elemento desta lista. Mostre a avaliação desta chamada, para o parâmetro L = List(„z","k","ya","b"). Veja as observações abaixo.

Observações:
- Exemplo de funções pré-definidas em linguagens de programação funcional que também poderão ser usadas: *reduceLeft, reduceRight, filter, forAll, exists, forEach, foldLeft, combines*. Não é necessário implementar uma chamada recursiva

Questão 4 : (20 pontos)
A função abaixo retorna o maior valor entre um parametro inteiro **m** e os elementos de uma lista de inteiros **l.** Entretanto, a implementação abaixo não está de acordo com o paradigma de programação funcionalista. Crie uma nova versão da função abaixo, com o mesmo comportamento, porém seguindo o paradigma de programação funcionalista.

```
def maior (m: Int, l: List[Int]):Int =
if (l.isEmpty)
  m
else {
  var aux=0;
  if (l.head > m )
    aux = l.head;
  else
    aux = m;
   maior(aux,l.tail);
  }
```