Lista de Exercícios 07
Programação Funcional: SML Avançado

Exercício 01) Resolva os exercícios a seguir usando as funções `map`, `foldr` e `foldl` em apenas uma linha, os famosos *one-liners*. Você pode usar outras funções, como as pré-definidas em SML, mas não pode escrever nenhuma função adicional, tampouco usar recursão explícita. Caso você precise de funções auxiliares, use funções anônimas. Por exemplo, se o problema for "*escreva uma função `add2` que recebe uma lista de inteiros e retorne uma lista com o número dois somada a cada elemento*", uma solução possível seria a seguinte função: `fun add2 x = map (fn a => a + 2) x;`

a) A função `il2rl`, de tipo `int list -> real list`, que recebe uma lista de inteiros e retorne uma lista com cada valor convertido para real. Por exemplo, a chamada `il2rl [1,2,3]` deveria retornar `[1.0,2.0,3.0]`.
b) A função `ordList`, de tipo `char list -> int list`, que recebe uma lista de caracteres e retorne uma lista com os códigos ASCII desses caracteres. Por exemplo, a chamada `ordList [#"A",#"b",#"C"]` deveria retornar `[65,98,67]`.
c) A função `squareList`, de tipo `int list -> int list`, que recebe uma lista de inteiros e retorne uma lista dos quadrados destes inteiros. Por exemplo, a chamada `squareList [1,2,3,4]` deveria retornar `[1,4,9,16]`.
d) A função `multPairs`, de tipo `(int * int) list -> int list`, que recebe uma lista de pares de inteiros e retorne uma lista com os produtos de cada par. Por exemplo, a chamada `multPairs [(1, 2), (3, 4)]` deveria retornar `[2, 12]`.
e) A função `incList`, de tipo `int list -> int -> int list`, que recebe uma lista de inteiros e um inteiro que será usado como incremento. A função deve retornar a mesma lista de entrada, exceto que o incremento será somado a cada elemento da entrada. Por exemplo, a chamada `incList [1,2,3,4] 10` deveria retornar `[11,12,13,14]`. Note que essa função é currificada.
f) A função `sqSum`, de tipo `int list -> int`, que recebe uma lista de inteiros e retorne a soma dos quadrados destes inteiros. Por exemplo, a chamada `sqSum [1,2,3,4]` deveria retornar `30`.
g) A função `bor`, de tipo `bool list -> bool`, que recebe uma lista de booleanos e retorne o *ou lógico* de todos eles. Se a lista estiver vazia, então deve-se retornar `false`.
h) A função `dupList`, de tipo `'a list -> 'a list`, cujo resultado seja a lista de entrada com cada elemento repetido em sequência. Por exemplo, a chamada `dupList [1,2,3]` deveria retornar `[1,1,2,2,3,3]`. Para a lista vazia, deve-se retornar uma lista vazia.
i) A função `myLength`, de tipo `'a list -> int`, que retorne o tamanho da lista de entrada. Você não pode usar a função predefinida `length`.

j) A função `is2absrl`, de tipo `int list -> real list`, que recebe uma lista de inteiros e retorne uma lista contendo o valor absoluto de cada um desses inteiros convertidos em números reais.
k) A função `trueCount`, de tipo `bool list -> int`, que recebe uma lista de valores booleanos e retorne o número de valores verdadeiros nesta lista.
l) A função `maxPairs`, de tipo `(int * int) list -> int list`, que recebe uma lista de pares de inteiros e retorne a lista formada pelos maiores elementos de cada par. Por exemplo, a chamada `maxPairs [(1,3),(4,2),(3,4)]` deveria retornar a lista `[3,4,4]`.
m) A função `lconcat`, de tipo `'a list list -> 'a list`, que recebe uma lista de listas como entrada e retorne uma lista formada pela composição das listas de entrada, em ordem. Por exemplo, a chamada `lconcat [[1,2],[3,4,5,6],[7]]` deveria retornar `[1,2,3,4,5,6,7]`. Note que você não pode usar função predefinida chamada `concat` para isso.
n) A função `min`, de tipo `int list -> int`, que retorne o menor inteiro dentre os números da lista de entrada. A sua função pode gerar um erro se a lista de entrada estiver vazia.
o) A função `member`, de tipo `''a * ''a list -> bool`, tal que a chamada `member (e, L)` seja verdadeira se, e somente se, o elemento `e` pertence a lista `L`.
p) A função `append`, de tipo `'a list -> 'a list -> 'a list`, que recebe duas listas e retorne a lista resultante da concatenação da segunda sobre a primeira. Por exemplo, a chamada `append [1,2,3] [4,5,6]` deveria retornar `[1,2,3,4,5,6]`. Note que essa função é currificada. Não use a função predefinida `concat`, tampouco use o operador `@`.
q) A função `less`, de tipo `int * int list -> int list`, tal que `less (e, L)` retorne uma lista com todos os elementos de `L` que sejam menores que `e`.
r) A função `evens`, de tipo `int list -> int list`, que recebe uma lista de inteiros e retorne a lista de todos os inteiros pares dessa lista. Por exemplo, a chamada `evens [1,2,3,4]` deveria retornar `[2,4]`.
s) A função `convert`, de tipo `('a * 'b) list -> 'a list * 'b list`, que converte uma lista de pares em um par de listas. Por exemplo, a chamada convert `[(1,2), (3,4), (5,6)]` deveria retornar o par de listas `([1,3,5],[2, 4, 6])`.

Exercício 02) As questões a seguir referem-se ao conceito de tipos de dados.

a) Crie o tipo de dados `suit`, cujos valores sejam os quatro naipes de um baralho: `Hearts`, `Clubs`, `Diamonds` e `Spades`. Depois, usando essa definição, faça a função `suitname`, de tipo `suit -> string`, que retorne um valor do tipo *string* descrevendo o nome do naipe.
b) Crie o tipo de dados `number`, cujos valores sejam números inteiros ou reais. Depois, usando essa definição, faça a função `plus`, de tipo `number -> number -> number`, que some dois números. A função deve converter inteiros para reais quando receber um parâmetro real e inteiro.

c) Considere o tipo de dados: `datatype intnest = INT of int | LIST of intnest list`. Faça a função `addUp`, de tipo `intnest -> int`, que some todos os inteiros em um `intnest`.

d) Considere o tipo de dados `datatype 'element mylist = NIL | CONS of 'element * 'element mylist` para representar uma lista de elementos. Usando essa definição, faça:

   i. A função `prod`, de tipo `mylist -> int`, que receba uma lista `x`, de tipo `int mylist`, e retorne o produto de todos os elementos de `x`. Se a lista for `NIL`, a função deverá retornar o número `1`.

   ii. A função `append`, de tipo `'a mylist -> 'a mylist -> 'a mylist`, que recebe dois valores do tipo `mylist`, por exemplo, `a` e `b`, e retorne um valor do tipo `mylist` contendo todos os elementos de `a` seguidos de todos os elementos de `b`.

   iii. A função `reverse`, de tipo `'a mylist -> 'a mylist`, que recebe uma lista `a`, de tipo `mylist`, e retorne uma lista, de tipo `mylist`, com todos os elementos de `a` em ordem inversa.

e) Considere o tipo de dados `datatype 'data tree = Empty | Node of 'data tree * 'data * 'data tree` para representar uma árvore binária. Usando essa definição, faça:

   i. A função `revTree`, de tipo `'a tree -> 'a tree`, que inverte o conteúdo de uma árvore. Por exemplo:

```
-revTree (Node(Node(Empty,1,Empty),2,Node(Empty,3,Empty)));
val it = Node(Node(Empty,3,Empty),2,Node(Empty,1, Empty)) : int tree
```

   ii. A função `appendall`, de tipo `'a list tree -> 'a list`, que recebe uma árvore de listas e retorne a lista resultante da concatenação de todas as listas na árvore. Coloque as listas juntas em uma viagem em ordem (*in-order*) na árvore, isto é, primeiro todas as listas da parte esquerda de um nó, então a lista do próprio nó, e finalmente todas as listas à direita do nó.

   iii. A função `isComplete`, de tipo `'a tree -> bool`, que verifique se uma árvore binária é completa, ou seja, se cada nó tem dois filhos ou nenhum filho. Em termos do tipo de dados `tree`, a árvore é completa se cada nó tem dois filhos do tipo `Empty` ou dois filhos do tipo `Node`, mas não um de cada.

   iv. A função `makeBST`, de tipo `'a list -> ('a * 'a -> bool) -> 'a tree`, que organize os itens da lista em uma árvore de busca binária. Uma árvore de busca binária é uma árvore binária com algumas propriedades especiais. Ela pode ser vazia (`Empty`) ou pode ser um nó (`Node`) contendo uma árvore à esquerda, um elemento `x` e uma árvore à direita. Neste caso, todos os dados na árvore precisam ser diferentes e todos os itens na árvore à esquerda precisam ser menores que `x`, e todos os itens à direita precisam ser maiores que `x`. Obviamente as sub-árvores também precisam ser árvores binárias de busca. A propriedade de comparação (menor no exemplo) deve ser passada à função. Sua árvore não precisa ser balanceada e você pode assumir que a lista de entrada contém somente elementos distintos.

v. A função `searchBST`, de tipo `''a tree -> (''a * ''a -> bool) -> ''a -> bool`, que busque um elemento em uma árvore de busca binária. A busca deve ser feita em `log(n)`, ou seja, não se deve fazer uma busca completa na árvore.

Exercício 03) Considere o tipo de dados `exp` a seguir que descreve uma linguagem muito simples para realização de operações aritméticas. Ela suporta valores numéricos (`NUM`) com as operações de adição (`SUM`), multiplicação (`MUL`) e negação aritmética (`NEG`) que inverte o sinal de um número.

```
datatype exp = NUM of int |
               SUM of exp * exp |
               MUL of exp * exp |
               NEG of exp;
```

Crie a função `interpret`, de tipo `exp -> int`, que interprete expressões nessa linguagem. Essa função deve assumir a semântica tradicional das operações aritméticas. Por exemplo, a expressão dada por `val x = SUM((MUL(NUM 2,NUM 3)),SUM(NEG(NUM 3),NUM 5))` corresponde à expressão aritmética `(2 × 3) + ((−3) + 5)`. Portanto, a chamada `interpret x` deve retornar o valor inteiro 8.