Lista de Exercícios 03
Programação Funcional: SML Básico

Exercício 01) Escreva funções simples em SML, sem usar casamento de padrões, para cada um dos problemas a seguir.

a) A função `cube`, de tipo `int -> int`, que retorne o cubo de sua entrada.
b) A função `cuber`, de tipo `real -> real`, que retorne o cubo de sua entrada.
c) A função `forth`, de tipo `'a list -> 'a`, que retorne o quarto elemento da lista passada como entrada. Não precisa verificar se essa lista tem menos de quatro elementos.
d) A função `min3`, de tipo `int * int * int -> int`, que retorne o menor elemento de uma tupla de três inteiros.
e) A função `rem2`, de tipo `'a * 'b * 'c -> 'a * 'c`, que converte uma tupla com três elementos em uma tupla com dois elementos sem o segundo elemento da lista de entrada.
f) A função `thirds`, de tipo `string -> char`, que retorne o terceiro caractere de uma *string*. Não precisa verificar se essa *string* tem pelo menos três caracteres.
g) A função `cycle1`, de tipo `'a list -> 'a list`, que recebe uma lista l_e de entrada e retorne uma lista l_s de saída, tal que o primeiro elemento de l_e tenha sido movido para a última posição de l_s. Por exemplo, `cycle1 [1,2,3,4]` retorna `[2,3,4,1]`.
h) A função `sort3`, de tipo `real * real * real -> real list`, que retorne uma lista ordenada com os três números passados na entrada.
i) A função `del3`, de tipo `'a list -> 'a list`, cuja lista de saída seja a mesma que a lista de entrada, mas com seu terceiro elemento removido. Não precisa verificar se essa lista tem menos de três elementos.

Exercício 02) Escreva funções em SML, desta vez usando casamento de padrões, para cada um dos problemas a seguir.

a) A função `pow`, de tipo `real * int -> real`, que eleve um número real a uma potência inteira. Não é preciso verificar potências negativas.
b) A função `sqsum`, de tipo `int -> int`, que recebe um número não negativo `n` e retorne a soma de todos os quadrados de `0` até `n`. Não precisa verificar se `n` é menor que `0`.
c) A função `less`, de tipo `int * int list -> int list`, tal que `less (e, L)` retorne uma lista formada por todos os inteiros de `L` menores que `e`.
d) A função `repeats`, de tipo `''a list -> bool`, tal que `repeats L` retorne verdadeiro se, e somente se, a lista `L` possui dois elementos consecutivos iguais.
e) A função `member`, de tipo `''a * ''a list -> bool`, de modo que `member (e, L)` retorne verdadeiro se, e somente se, `e` for um elemento da lista `L`.

f) A função `cyclen`, de tipo `'a list * int -> 'a list`, que recebe uma lista `l` e um inteiro `n` e retorne uma cópia da lista `l`, mas com os `n` primeiros elementos movidos para o final da lista. Por exemplo, `cyclen ([1,2,3,4,5,6],2)` retorna `[3,4,5,6,1,2]`.
g) A função `max`, de tipo `int list -> int`, que retorne o maior elemento de uma lista de inteiros. Não é quando a lista é vazia. Dica: use uma função auxiliar `maxhelper`, de tipo `int list * int -> int`, que recebe como segundo parâmetro o maior elemento visto até o momento.
h) A função `isPrime`, de tipo `int -> bool`, que retorne verdadeiro se a entrada é um número inteiro primo e falso caso contrário. Não é preciso tratar o caso em que a entrada é negativa.
i) A função `select`, de tipo `'a list * ('a -> bool) -> 'a list`, que recebe uma lista `l` e uma função `f` e aplica `f` em todos os elementos de `l`. Por exemplo, `select ([1,2,3,4,5,6,7,8,9,10],isPrime)` retorna `[2,3,5,7]`.