Lista de Linguagens de Programação – 9

Nome: _____ Matrícula: _____

A solução esperada para estes 25 próximos exercícios deve ter somente uma linha, e deve usar as funções `map`, `foldr`, e `foldl`. É claro que você poderia usar outras funções, principalmente aquelas que já são predefinidas em SML/NJ, mas *não escreva nenhuma função adicional, tampouco use recursão explícita.* Se você precisar de funções auxiliares, use funções anônimas. Por exemplo, se o problema fosse "escreva uma função `add2` que receba uma lista de inteiros e retorne uma lista com o número dois somado a cada elemento", a sua resposta poderia ser:

```
fun add2 x = map (fn a => a + 2) x;
```

Você já viu alguns destes problemas antes. O ponto agora é resolvê-los desta nova maneira, elegante e concisa.

1. Escreva uma função `il2rl`, de tipo `int list -> real list` que receba 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]`.

2. Escreva uma função `ordList`, de tipo `char list -> int list` que receba uma lista de caracteres e retorne uma lista com os códigos ASCII destes caracteres. Por exemplo, a chamada `ordList [#"A", #"b", #"C"]` deveria retornar `[65, 98, 67]`.

3. Escreva uma função `squareList`, de tipo `int list -> int list` que receba 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]`.

4. Escreva uma função `multPairs`, de tipo `(int * int) list -> int list` que receba 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]`.

5. Escreva uma função `incList`, de tipo `int list -> int -> int list` que receba 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 de entrada. Por exemplo, a chamada `incList [1, 2, 3, 4] 10` deveria retornar `[11, 12, 13, 14]`. Note que a função é currificada.

6. Escreva uma função `sqSum`, de tipo `int list -> int` que receba uma lista de inteiros e retorne a soma dos quadrados destes inteiros. Por exemplo, a chamada `sqSum [1, 2, 3, 4]` deveria retornar `30`.

7. Escreva uma função `bor`, de tipo `bool list -> bool` que receba uma lista de booleanos e retorne o *or* lógico de todos eles. Se a lista estiver vazia, então `false` deve ser retornado.

8. Escreva uma função `band`, de tipo `bool list -> bool` que receba uma lista de booleanos e retorne o *and* lógico de todos eles. Se a lista estiver vazia, então `true` deve ser retornado.

9. Escreva uma função `bxor`, de tipo `bool list -> bool` que receba uma lista de booleanos e retorne o *or* exclusivo de todos eles. Isto quer dizer que esta função deveria retornar `true` se o número de valores `true` na lista for ímpar, e `false` caso contrário. Se a lista estiver vazia, então `false` deve ser retornado.

10. Escreva uma 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]`. Se a lista de entrada é nula, o resultado deve ser a lista nula.

11. Escreva uma função `myLength`, de tipo `'a list -> int` que retorne o tamanho da lista de entrada. Obviamente você **não** pode usar a função predefinida `length` para resolver este exercício.

12. Escreva uma função `is2absrl`, de tipo `int list -> real list` que receba uma lista de inteiros e retorne uma lista contendo o valor absoluto de cada um destes inteiros, convertido em um número real.

13. Escreva uma função `trueCount`, de tipo `bool list -> int` que receba uma lista de valores booleanos e retorne o número de valores verdadeiros nesta lista.

14. Escreva uma função `maxPairs`, de tipo `(int * int) list -> int list` que receba 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]`.

15. Escreva uma função `myImplode` que funcione como a função `implode`, predefinida em SML/NJ. Em outras palavras, esta função deve ter o tipo `char list -> string`. A função recebe uma lista de caracteres e retorna uma *string* contendo estes mesmos caracteres, na mesma ordem.

16. Escreva uma função `lconcat`, de tipo `'a list list -> 'a list` que receba 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 existe uma função predefinida, chamada `concat`, que faz exatamente isto. Obviamente você **não** pode usar esta função.

17. Escreva uma função `max`, de tipo `int list -> int` que retorne o maior inteiro dentre os números da lista de entrada. A sua função pode gerar um erro se acaso a lista de entrada estiver vazia.

18. Escreva uma 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 acaso a lista de entrada estiver vazia.

19. Escreva uma 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`.

20. Escreva uma função `append`, de tipo `'a list -> 'a list -> 'a list` que receba 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]`. Não use a função predefinida `concat`, tampouco use o operador `@`.

21. Escreva uma 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`.

22. Escreva uma função `evens`, de tipo `int list -> int list` que receba uma lista de inteiros e retorne a lista de todos os inteiros pares da lista original. Por exemplo, a chamada `evens [1, 2, 3, 4]` deveria retornar `[2, 4]`.

23. Escreva uma função `convert`, de tipo `('a * 'b) list -> 'a list * 'b list` que converta uma lista de pares em um par de listas. Por exemplo, a chamada `convert [(1, 2), (3, 4), (5, 6)]` deveria retornar a lista `[1, 3, 5], [2, 4, 6]`.

24. Escreva uma função `myMap`, que possua o mesmo tipo e a mesma semântica de `map`. Claro que você **não** pode usar a função predefinida `map`.

25. Neste exercício, um polinômio será representado usando uma lista com os seus coeficientes reais, começando com a constante, e tendo como último elemento o coeficiente de grau mais alto. Por exemplo, $3x^2 + 5x + 1$ seria representado como a lista `[1.0, 5.0, 3.0]`. Já $x^3 - 2x$ seria representado como `[0.0, 2.0, 0.0, 1.0]`. Escreva uma função `eval`, de tipo `real list -> real -> real` que receba um polinômio representado desta forma, mais um valor de x, e retorne o valor do polinômio para aquele dado x. Por exemplo, `eval([1.0, 5.0, 3.0], 2.0)` deveria produzir o resultado `23.0`, já que, se $x = 2$, então $3x^2 + 5x + 1 = 23$. Este exercício já apareceu na lista 6, exceto que agora o tipo de função é currificado, e a resposta deve ser escrita com uma linha somente.