Lista 04

Programação Funcional

Prof. Maycon Amaro

Orientações

- Apenas o primeiro exercício é em C. Os outros são em Haskell.
- Ao criar o projeto dos exercícios dessa lista, deixe o `main` imprimindo `"Not yet"` como mostrado no exemplo *Alternativa* na aula 03 e use o `ghci` para executar suas funções.
- Use o `ghci` para consultar tipos e realizar alguns testes simples envolvendo funções e operadores do prelúdio para entender como eles funcionam.
- Se quiser feedback sobre suas soluções, envie para `maycon.amaro@ufop.edu.br`, iniciando o assunto com [BCC222].

Exercícios

1. Em C, implemente as duas versões de fibonacci apresentadas e compare o tempo que demora para cada uma computar o 50^{o} termo da fibonacci.
2. Implemente a função `concatenar` com tipo `[Int] -> [Int] -> [Int]`, que concatena duas listas de inteiros. Sua implementação deve ter o mesmo comportamento que o operador `++` do prelúdio.
3. Implemente a função `reverter1` com tipo `[Int] -> [Int]` que inverte a ordem dos elementos de uma lista. Sua implementação deve usar recursão explícita e o operador de concatenação `++`, e deve ter o mesmo comportamento que a função `reverse` do prelúdio.
4. Implemente uma outra versão, `reverter2`, que usa `foldr` e não usa o operador `++`. Compare `reverter1` e `reverter2`: qual delas parece mais eficiente?
5. Implemente a função `primeiro` com tipo `[Int] -> Int` que retorna o primeiro elemento de uma lista. Se a lista for vazia, retorne `error "empty list"`. Verifique que o comportamento da sua função se parece com a `head` do prelúdio.
6. Implemente a função `coletar` com tipo `Int -> [Int] -> [Int]` que retorna os n primeiros elementos da lista. Sua função deve ter o mesmo comportamento de `take` do prelúdio.

7. Implemente a função de ordem superior `todos` com tipo `(Int -> Bool) -> [Int] -> Bool` que retorna verdadeiro se todos os elementos da lista satisfazem a função fornecida e falso caso contrário. Sua função deve ter o mesmo comportamento de `all` do prelúdio. Teste simples: `all even [2, 4, 6]`.
8. Implemente a função `positivar` com tipo `[Int] -> [Int]` que troca o sinal dos elementos negativos e elimina as ocorrências de 0.
9. Implemente a função `replicar` com tipo `Int -> Int -> [Int]` que cria uma lista em que todos os elementos são iguais. O segundo parâmetro indica o elemento e o primeiro indica a quantidade de repetições. Sua função deve ter o mesmo comportamento de `replicate` do prelúdio.
10. Implemente a função `quantidade` com tipo `[Int] -> Int` que conta o número de elementos de uma lista, usan Sua função deve ter o mesmo comportamento de `length` do prelúdio. Tente usar funções de ordem superior ao invés de recursão explícita.