Listas por compreensão

Na matemática é costume definir conjuntos por compreensão à custa de outros conjuntos. Por exemplo:
{ 2x | x ∈ {10,3,7,2}} --> O conjunto {20,6,14,4}
{ n | n ∈ {4,-5,8,20,-7,1} ∧ 0 <= n <= 10 } --> O conjunto {4,8,1}

Em haskell podem definir-se listas por compreensão, de modo semelhante, contruindo novas listas às custas de outras listas.
`[2*x | x <- [10,3,7,2]]` --> A lista `[20,6,14,4]`
`[n | n <- [4,-5,8,20,-7,1], 0 <= n <= 10]` --> A lista `[4,8,1]`

-> A expressão `n <- [4,-5,8,20,-7,1]` é chamada de gerador da lista.
-> A expressão `0 <= n <= 10` é uma guarda que restringe os valores produzidos pelo gerador que a precede.

Assim, as listas por compreensão podem ter vários geradores e várias guardas, sendo que mudar a ordem dos geradores muda a
ordem dos elementos na lista final. Por exemplo:

```
[(x,y) | x <- [1,2,3], y <- [4,6]]
        --> [(1,4),(1,6),(2,4),(2,6),(3,4),(3,6)]
[(x,y) | y <- [4,6], x <- [1,2,3]]
        --> [(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)]
```

Uso da notação (..) :

```
> [1..5]
[1,2,3,4,5]
```

```
> [1,10..100]
[1,10,19,28,37,46,55,64,73,82,91,100]
```

```
> [20,15..(-7)]
[20,15,10,5,0,-5]
```

```
> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
```

Com esta notação é possível definir listas infinitas.

Funções e Listas por Compreensão

Podem-se definir funções usando listas por compreensão.
Exemplo, a função de ordenação de listas quisort:

```
qsort :: (Ord a) => [a] -> [a]
qsort [] = []
qsort (x:xs) = (qsort [y | y <- xs, y < x]) ++[x]++ (qsort [y | y <- xs, y >= x])
```

Exemplo, usando a função zip e listas por compreensão, podemos definir a função que calcula a lista de posições de um dado valor
numa lista:

```
posicoes :: Eq a => a -> [a] -> [Int]
posicoes x l = [i | (y,i) <- zip l [0..], x == y]
```

-> O lado esquerdo do gerador da lista é um padrão.

Exemplo, calcular os divisores de um número positivo:

```haskell
divisores :: Integer -> [Integer]
divisores n = [x | x <- [1..n], n `mod` x == 0]
```

#haskell
#SoftwareEngineering