Universidade Federal do Ceará	Campus de Quixadá
Bacherelado em Ciência da Computação	
Programação Funcional	
1° Avaliação Parcial	Data 06/05/2016
Nome:	

1. (2.0) Defina cada uma das seguintes funções usando apenas funções pré-definidas no Prelúdio:

   (a) Escreva a função `dividePos :: Int -> [a] -> ([a],[a])` tal que (dividePos n xs) devolve uma tupla onde o primeiro elemento é um prefixo de xs de tamanho n e o segundo elemento é o restante da lista. Por exemplo:

```
dividePos 6 "Hello World!" == ("Hello ","World!")
dividePos 3 [1,2,3,4,5] == ([1,2,3],[4,5])
dividePos 1 [1,2,3] == ([1],[2,3])
dividePos 3 [1,2,3] == ([1,2,3],[])
dividePos 4 [1,2,3] == ([1,2,3],[])
dividePos 0 [1,2,3] == ([],[1,2,3])
dividePos (-1) [1,2,3] == ([],[1,2,3])
```

   Dica: use a função take e drop.

   (b) Escreva a função `dividePred :: (a -> Bool) -> [a] -> ([a],[a])` tal que (dividePred p xs) devolve uma tupla onde o primeiro elemento é a maior prefixo de xs (possivelmente vazio) de elementos que satisfazem p e o segundo elemento é o restante da lista.

```
dividePred (<3) [1,2,3,4,1,2,3,4] == ([1,2],[3,4,1,2,3,4])
dividePred (<9) [1,2,3] == ([1,2,3],[])
dividePred (<0) [1,2,3] == ([],[1,2,3])
```

   Dica: use a função takeWhile e dropWhile.

2. (2.0) Defina as seguintes funções usando compreensão de listas:

   (a) Escreva a função `prefixos :: [a] -> [a]` usando compreensão de listas tal que prefixos xs devolve uma lista contendo todos os prefixos de xs.

```
prefixos "abab" == ["", "a", "ab", "aba", "abab"]
prefixos [1,2] == [ [], [1], [1,2]]
prefixos [1,2,3,4] == [ [], [1], [1,2], [1,2,3], [1,2,3,4] ]
```

   (b) Escreva uma função `filtrandoCaudas :: [[Int]] -> [[Int]]` usando compreensão de listas tal que (caudas xss) devolve uma lista contendo a cauda das listas não vazias, onde a cabeça da lista é maior que 5.

```
filtrandoCaudas [ [7,6,3],[],[6,4,2],[9,4,3],[5,5,5]] == [[6,3],[4,2],[4,3]]
filtrandoCaudas [ [5,6,3],[],[9,4,2],[3,4,3],[8,5,5]] == [[4,2],[5,5]]
```

   Dica : use as funções head, tail, null.

3. (2.0)O método de ordenação por seleção é um método simples que pode ser separado em três passos:

   i Encontrar o menor elemento de uma lista, pode ser encontrado utilizando a função `minimum :: Ord a => [a] -> a`.

   ii Obtendo uma nova lista removendo o menor elemento da lista original.

   iii A lista ordenada final $(x : xs)$, onde x é o menor elemento e xs é uma lista obtida pela ordenação da lista do passo anterior.

   (a) Defina a função `remove :: (Ord a) => a -> [a] -> [a]` tal que (remove x xs) retorna uma lista obtida removendo uma ocorrência de x.

```
remove 2 [1,2,3,4] == [1,3,4]
remove 4 [1,2,4,2,4] == [1,2,2,4]
```

(b) Usando a função minimum e remove, escreva a função `ordsel :: Ord a => [a] -> [a]` tal que (`ordsel xs`) recebe uma lista possivelmente não ordenada xs e devolve uma lista ordenada implementando a ordenação por seleção.

```
ordsel [2,1,4,3] == [1,2,3,4]
ordsel [1,4,2,3] == [1,2,3,4]
ordsel [1,5,3,4] == [1,3,4,5]
```

4. (2.0) Defina as seguintes funções usando recursão:

   (a) Defina recursivamente a função `descompacta :: :: [(a, b)] -> ([a], [b])` que transforma uma lista de pares ordenado em um par ordenado onde o primeiro elemento é uma lista dos primeiros componentes dos pares ordenados e o segundo elemento é uma lista dos segundos componentes dos pares ordenados.

```
descompacta [(1,2),(3,4),(5,6),(4,5)] == ([1,3,5,4],[2,4,6,5])
descompacta [(1,2),(3,4),(5,6),(4,5),(5,6)] == ([1,3,5,4,5],[2,4,6,5,6])
```

   (b) Defina recursivamente a função `ordenada :: Ord a => [a] -> Bool` tal (*ordenada xs*) verifica se xs é uma lista ordenada. Por exemplo,

```
ordenada [2,3,5] == True
ordenada [3,2,5] == False
```

5. (2.0) Considere que uma lista é uma subsequência de uma outra lista se os elementos da primeira ocorrem na segunda na mesma ordem. Por exemplo, "abcd"é uma subsequência "XYabZwcd". Uma lista é uma sublista de uma outra se os elementos da primeira ocorrem na segunda, de maneira contígua. Por exemplo, "abcd"é uma sublista de "XYabcd".

   (a) Escreva a função `subSequencia :: [a] -> [a] -> Bool` tal que subsequencia xs ys decide se xs é uma subsequencia de ys.

   (b) Escreva a função `subLista :: [a] -> [a] -> Bool` tal que subLista xs ys decide se xs é uma sublista de ys.