1ª Lista de Exercícios de Paradigmas de Linguagens Computacionais

Professor: Fernando Castor

Monitores:

Paulo Barros <pbsf>,

Leonardo Brayner <lbs2>,

Cleivson Siqueira de Arruda <cleivson.tb@gmail.com>,

Irineu Moura <imlm2>,

Caio Sabino Silva <ccss2>,

CIn-UFPE - 2010.2

Disponível desde: 10/09/2010

Entrega: 24/09/2010

A lista deverá ser respondida **em dupla**. A falha em entregar a lista até a data estipulada implicará na perda de 0,25 ponto na **média** da disciplina para os membros da dupla. Considera-se que uma lista na qual **menos que 30%** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 80%** das questões corretamente respondidas implica em um acréscimo de 0,125 ponto na média da disciplina para os membros da dupla. Se **qualquer situação de cópia de respostas** for identificada, os membros **de todas as duplas envolvidas** perderão 0,5 ponto na média da disciplina. O mesmo vale para respostas obtidas a partir da Internet. As respostas deverão ser entregues **exclusivamente em formato texto ASCII** (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas para o monitor responsável por sua dupla, **sem** cópia para o professor. Tanto podem ser organizadas em arquivos separados, um por questão (e, neste caso, deverão ser zipadas), quanto em um único arquivo texto onde a resposta de cada questão está devidamente identificada e é auto-contida (uma parte da resposta de uma questão que seja útil para responder uma outra deve estar duplicada neste última). Para definir qual será o monitor responsável por corrigir a sua lista, vá até o endereço http://sites.google.com/a/cin.ufpe.br/monitoria-plc/ e inclua (a página é editável) os nomes dos dois membros da sua dupla, junto com os logins, embaixo do nome do monitor que tiver menos duplas e com base na ordem em que os nomes dos monitores aparecem na página.

1) Defina as funções lógicas `and`, `or`, `nand`, `nor`, `xor`, `impl` (implicação) e `eq` (equivalência) para pares de valores booleanos.

Exemplo

```
x :: Bool
x = True

y :: Bool
y = False

Prelude> and (x,or(x,y))
Prelude> True
```

2) Defina uma função `inserirElemento :: [a] -> a -> Int -> [a]`, que recebe uma lista de elementos, um elemento e um inteiro, inserindo o elemento recebido na posição dada pelo inteiro, retornando a lista final. Considere que 1 é a primeira posição e não serão passadas entradas com a posição menor do que 1.

Exemplo

```
Prelude> inserirElemento ['a','b','c']  'x'  2
Prelude> ['a','x','b','c']
```

3) Defina uma função `eliminarRepetidos :: [Int] -> [Int]` , que recebe uma lista de inteiros e devolve a mesma lista, mas sem elementos repetidos, mantendo apenas a primeira ocorrência de cada elemento.

Exemplo

```
Prelude> eliminarRepetidos [1,2,1,2,3,4,5,3,7]
Prelude> [1,2,3,4,5,7]
```

4) Se `id` é a função identidade polimórfica definida como `id x = x`, explique o comportamento das expressões:

`(id . f)` `(f . id)` `id f`

Se `f` é do tipo `Int -> Bool`, em que instância de seu tipo genérico `a -> a` `id` é usada em cada caso?

Exemplo

`Int -> Int` e `Bool -> Bool` são instancias do tipo genérico `a -> a`.

5) Defina uma função `inverterOrdem :: String -> String` , que recebe uma *string* com uma frase de palavras separadas pelo caractere `' '` e retorna tal frase com a ordem de palavras invertida.

Exemplo

```
Prelude> inverterOrdem “Romeu e Julieta”
Prelude> “Julieta e Romeu”
```

6) Defina uma função `fatoresPrimos :: Int -> [Int]` , que recebe um inteiro positivo e retorna uma lista de inteiros contendo os fatores primos do inteiro recebido. Essa lista deve estar ordenada de forma ascendente.

Exemplo

```
Prelude> fatoresPrimos 315
Prelude> [3,3,5,7]
```

7) Defina uma função `subst :: String -> String -> String -> String` tal que `subst velha nova st` resulte numa nova *string* em que a primeira ocorrência de `velha` foi substituída por `nova`.

Exemplo

```
Prelude> subst "quarta" "quinta" "Ontem foi uma quarta-feira."
Prelude> "Ontem foi uma quinta-feira."
```

8) Defina a função `split :: [a] -> a -> [[a]]` que recebe uma lista e divide a lista em cada ocorrência do elemento passado como parâmetro, retornando uma lista das divisões da lista original.

Exemplo

```
Prelude> split “Cada palavra vai estar numa lista diferente” “ ”
Prelude> [“Cada”, “palavra”, “vai”, “estar”, “numa”, “lista”, “diferente”]
```

9) Defina a função `concat :: [String] -> String -> String` que recebe uma lista de *strings* e concatena todas inserindo a *string* do parâmetro entre elas e retorna a *string* concatenada.

Exemplo

```
Prelude> concat [“As”, “palavras”, “juntas”] “ ”
Prelude> “As palavras juntas”
```

10) Defina a função `juntarEMapear` que recebe como parâmetro duas funções e duas listas (nesta ordem) e retorna como resultado o mapeamento da segunda função parâmetro sobre a junção das duas listas.
A junção pode ser vista como a união de duas listas de tipos `a` e `b` (quaisquer) aplicando a cada par de elementos a primeira função parâmetro para resultar numa terceira lista de um tipo `c` qualquer, ou seja, a junção das listas `[1..5]` e `[-5..(-2)]` utilizando a função soma `(+)` retornaria a lista `[-4,-2,0,2]`.

Obs:
- A função deve ser o mais genérica possível;
- na junção de listas de tamanhos diferentes o tamanho da lista resultante é igual ao da menor lista;
- para realizar o mapeamento de uma função sobre uma lista a função `map` da biblioteca padrão pode ser utilizada.

Exemplo

```
Prelude> juntarEMapear (+) (abs) [1..10] [-10..(-1)]
Prelude> [9,7,5,3,1,1,3,5,7,9]
```