Lista de Exercícios 6 (Cap. 12) – INF05008

Siga as instruções sobre elaboração de exercícios de INF05008. Use o template fornecido.

Nesta lista, para todas as funções recursivas você deve incluir modelo da solução. Esse modelo, *em forma de comentários que podem ser colocados antes ou permeados ao código*, deve explicitar como o algoritmo funciona e deve ter o seguinte formato:

Modelo de algoritmo para listas

Dados *uma lista L e ...* → Dizer quais as entradas do algoritmo

se *< ...é o caso base da def. de lista... >* → Base: Identificar o caso trivial de listas e resolver o problema sem usar recursão

então *< ...resolver o problema ... >*

se *< ... não é o caso base da def. de lista... >* → Passo: Pode haver mais de um passo, e pode-se usar senão, se for o último caso

então *< ...combinar soluções... >*
< ...fazer algo com... > < o primeiro elemento da lista >
< ...solucionar o problema para... > < o resto da lista >

Atenção: No modelo da solução evite usar a palavra *recursão* (ou palavras derivadas desta): quando você sentir necessidade de dizer *e aplica a função recursivamente ao resto da lista*, diga o que essa aplicação deve devolver (por exemplo *a soma dos elementos do resto da lista, a imagem dos elementos do resto da lista, o menor dos números do resto da lista, ...*) e descreva como, a partir desse resultado (da aplicação recursiva), é construído o resultado da função. Note que dependendo do que o algoritmo deve fazer, pode ou não ser necessário combinar as soluções, usar o primeiro elemento da lista ou mesmo o resto da lista. O modelo acima deve ser adaptado em cada exercício. Alguns exemplos (lembre que o leitor deve conseguir entender como o algoritmo funciona lendo este modelo, pois ele é a descrição da solução):

TAMANHO

Obj: Determinar o tamanho de uma lista.
Dada *uma lista L*

se *a lista L estiver vazia*
então *devolver zero*
senão *somar um ao TAMANHO do resto da lista L*

SOMA DOS NEGATIVOS (versão 1)

Obj: Determinar a soma dos números negativos de uma lista.
Dada *uma lista de números L*

se *a lista L estiver vazia*
então *devolver zero*
se *o primeiro elemento da lista L for negativo*
então *somar o primeiro elemento da lista L com a SOMA DOS NEGATIVOS do resto da lista L*
senão *devolver a SOMA DOS NEGATIVOS do resto da lista L*

SOMA DOS NEGATIVOS (versão 2)

Obj: Determinar a soma dos números negativos de uma lista.
Dada *uma lista de números L*

se *a lista L estiver vazia*
então *devolver zero*
senão *somar*
o valor do primeiro elemento da lista L, se ele for menor que zero, senão somar zero, com a SOMA DOS NEGATIVOS do resto da lista L

Nesta lista, vamos trabalhar com os tipos de dados `Aluno` e `ListaAlunos`, definidos na aula. Você pode usar a função pré-definida `length`, que devolve o tamanho de uma lista (de qualquer tipo), além das funções `empty?`, `cons` e `list`. Não use outras funções pré-definidas sobre listas e nem funções de alta-ordem.

```
;; -----------------
;; TIPO ALUNO:
;; -----------------
(define-struct aluno (nome prova1 prova2 exerc))
;; Um elemento de Aluno é
;;    (make-aluno n p1 p2 e), onde
;;    n : String, é o nome do aluno
;;    p1: Número, é a nota da prova 1 do aluno
;;    p2: Número, é a nota da prova 2 do aluno
;;    e : Número, é a nota dos exercícios do aluno
;; -----------------
;; TIPO LISTAALUNOS:
;; -----------------
;; Uma ListaAlunos é
;; 1. empty, ou
;; 2. (cons a la), onde
;;     a : Aluno
;;     la: ListaAlunos
```

1. Construa a função `gera-lista-conceitos` que, dada uma lista de alunos, gera uma lista de pares contendo o nome e o conceito dos alunos. O conceito deve ser A, se a nota final do aluno for maior ou igual a 9, B, se estiver entre 7,5 (inclusive) e 9 (exclusive), C, se estiver entre 6 (inclusive) e 7,5 (exclusive), e D, caso contrário. Para definir o tipo do resultado, será necessário definir o tipo de dados que deve se chamar `ListaPar`, completando a definição a seguir:

```
;; -----------------
;; TIPO PAR:
;; -----------------
(define-struct par (nome conceito))
;; Um elemento de Par é
;;    (.....................), onde
;;    ..............  : String, é o nome do aluno
;;    ..............  : String, é o conceito do aluno
;; -----------------
;; TIPO LISTAPAR:
;; -----------------
;; Uma ListaPar é
;; 1. ............., ou
;; 2. (cons  .................), onde
;;     ......  : ..............
;;     .......: ..............
```

2. Desenvolva a função `devolve-conceitos` que, dada uma lista de pares nome/conceito, devolve a lista de conceitos da turma (note que aqui pode haver elementos repetidos, pois vários alunos podem ter ficado com o mesmo conceito).

3. Construa uma função chamada `filtra-nomes-conceito-A` que, dada uma lista de pares nome/conceito, devolve a lista de nomes de alunos que tiraram conceito A na turma.

4. Generalize função da questão anterior para filtrar a lista de pares recebendo também o conceito como entrada. A função deve se chamar `filtra-nomes-conceito`.

5. Defina a função `gera-resumo-turma` que, dada uma lista de alunos, devolve uma imagem com os nomes e conceitos dos alunos, sendo que os dados de cada aluno devem aparecer um ao lado do outro (veja alguns exemplos abaixo). Você pode escolher como mostrar os alunos, mas é necessário identificar claramente os alunos aprovados (com conceitos A, B ou C) e os reprovados (conceito D). Mostre, abaixo dos alunos, a distribuição percentual entre os conceitos na turma.

6. (Desafio - ponto extra) Construa a função `gera-resumo-turma-ordenada` que, dada uma lista de alunos, gera a imagem como na questão anterior, mas os alunos devem estar em ordem alfabética (considerando seus nomes). *Para fazer isso você pode adaptar a função* `ordena` *para considerar listas de pares ao invés de listas de números. Para comparar strings e verificar qual o menor considerando a ordem lexicográfica, pode-se usar a funcão* `string<=?`.