Lista de Exercícios 6 (Cap. 9) – 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, como na lista 4, vamos trabalhar com as cartas de UNO. Use as definições de dados a seguir para representar cartas de um baralho de UNO.

```
;; -----------------
;; TIPO CARTA:
;; -----------------
(define-struct carta (cor valor))
;; Um elemento do conjunto Carta é
;;   (make-carta c v)     onde
;;   c : String, é a cor da carta, que pode ser "azul", "verde", "amarelo", "vermelho" ou "preto" ou "livre"
;;   v : Número, é o valor da carta, que pode ser qualquer inteiro entre 0 e 9,ou um número negativo:
;;               -1 (PulaVez), -2 (Compra2), -3 (Inverte),-4 (Curinga-Compra4) ou -5 (Curinga)
```

1. Defina o tipo de dado `ListaDeCartas`, que contém todas as listas (finitas) de cartas de Uno e dê 4 exemplos de constantes deste tipo de dados. Defina também o tipo de dados `Jogador`, completando a definição a seguir. Para cada jogador deve ser guardado o nome do jogador (tipo String), a lista de cartas de sua mão e sua pontuação (tipo Número). Defina 2 constantes do tipo jogador. *Para pensar: Compare a definição da mão do jogador da lista 4 (usando uma estrutura) com a definição usando uma lista de cartas. Quais as vantagens/desvantagens de usar uma lista ao invés de uma estrutura?*

```
;; -----------------
;; TIPO JOGADOR:
;; -----------------
(define-struct jogador (nome mão pontos))
;; Um elemento do conjunto Jogador é
;;   (make-jogador ..............)     onde
;;   ...................
```

2. Desenvolva a função `desenha-cartas` que, dada uma lista de cartas, desenha todas as cartas desta lista lado a lado. Você pode usar a função `desenha-carta` do template, ou usar sua própria função (criada na lista 4) para desenhar cada carta.

3. A função `jogada-válida?` (ver template) verifica se, dadas uma carta da mão do jogador e uma da mesa, o jogador pode jogar esta carta. Usando esta função, construa uma função chamada `número-opções` que, dadas uma lista de cartas e uma carta da mesa, devolve o número de opções de cartas para jogar que existem nesta lista de cartas.

4. Construa uma função chamada `soma-pontos` que, dada uma lista de cartas, devolve a soma dos pontos das cartas desta lista. Segundo as regras do UNO, as cartas tem a seguinte pontuação:

   - as cartas numeradas valem o seu número;
   - as cartas "Compra2", "Inverte" e "PulaVez" valem 20;
   - as cartas "CuringaCompra4" e "Curinga" valem 50.

5. Construa a função `define-jogada` que, dados um jogador e a carta da mesa, nesta ordem, escolhe uma carta do jogador para ser jogada, de acordo com as regras do Uno. A carta a ser selecionada deve ser a primeira da lista de cartas da mão do jogador que for compatível com a carta da mesa. Caso o jogador não tenha nenhuma carta que possa ser jogada, a função deve retornar a mensagem `"Impossível jogar carta"`. *Atenção com o tipo do resultado: este programa pode devolver ou uma carta ou uma string! Para pensar: E se quiséssemos devolver a última encontrada ao invés da primeira, como fazer?*

6. (Desafio - ponto extra) Construa a função `mostra-jogadas-possíveis` que, dados a carta da mesa e um jogador, nesta ordem, gera uma imagem com o nome do jogador, a carta da mesa e todas as cartas da mão, identificando de alguma forma as cartas que poderiam ser jogadas e dizendo quantas opções o jogador tem. No caso de não poder ser jogada nenhuma carta, ao invés do número de opções, deve ser colocada a mensagem `Impossível jogar carta` na imagem resultante. Ver exemplos abaixo (você pode escolher uma forma de mostrar essas informações, não precisa ser igual a este exemplo). *Dica: Decomponha o problema em problemas menores e construa a solução através da composição das soluções dos problemas menores, preferencialmente reusando funções definidas nos exercícios anteriores, quando possível.*