Listas - Parte II

Fundamentos de Algoritmos

INF05008

Produzindo Listas

- Vimos funções que recebem listas como parâmetros
- Com isso, podíamos realizar cálculos sobre uma lista, tal como achar seu tamanho, encontrar a média de valores nela armazenados e verificar se a lista continha elementos de tipos ou valores específicos
- No entanto, funções também podem **produzir** listas

Funções que Produzem Listas

- **Exemplo:** Relembrando nosso problema de calcular o salário de um funcionário dado o número de horas que o mesmo trabalha

- **Problema:** Dada uma lista com valores de horas trabalhadas de cada funcionário, produzir uma lista dos salários correspondentes, calculados com base no valor de $12 por hora.

```
;; salário: number -> number
;; Calcular o salário com base em $12 por hora

(define (salário h)
  (* 12 h))
```

Funções que Produzem Listas (cont.)

```
;; horas->salário: lista-de-números -> lista-de-números
;; Cria uma lista de salários dada uma lista
;; das horas trabalhadas

(define (horas->salário ldn)...)
```

- Exemplos de entradas -> saídas correspondentes

```
empty                      ->                          empty
(cons 28 empty)            ->              (cons 336 empty)
(cons 40 (cons 28 empty)) -> (cons 480 (cons 336 empty))
```

Funções que Produzem Listas (cont.)

- Como toda **função recursiva**, pelo menos duas cláusulas aparecem no corpo da função:

```
(define (horas->salário ldn)
   (cond
      [(empty? ldn) ...]
      [else ... (first ldn) ...
                (horas->salário (rest ldn)) ...]))
```

Funções que Produzem Listas (cont.)

```
;; horas->salário: lista-de-números -> lista-de-números
;; Cria uma lista de salários dada uma lista das
;; horas trabalhadas

(define (horas->salário ldn)
   (cond
      [(empty? ldn) empty]
      [else (cons (salário (first ldn))
                  (horas->salário (rest alon)))]))

;; salário: number -> number
;; Calcula o salário considerando o pagamento de $12 por hora

(define (salário h)
   (* 12 h))
```

Exercício

Desenvolva uma função `elimina-caros` que elimina valores de brinquedos caros. A função recebe um número (*max*) e uma lista de preços de brinquedos *lpreço* e produz uma lista daqueles valores em *lpreço* que são iguais ou menores que *max*. Por exemplo:

```
(elimina-caros 1.0 (cons 2.95
                         (cons 0.95
                               (cons 1.0
                                     (cons 5 empty)))))
```

Valor esperado:

```
(cons 0.95 (cons 1.0 empty))
```

Lista de Dados Compostos

- Uma lista não necessariamente precisa conter dados atômicos
- Podemos ter **listas de estruturas**
- **Exemplo:** lista de dados compostos referentes aos brinquedos de uma loja

Lista de Estruturas

- Os brinquedos (bqd) de uma loja podem estar representados por um nome e um preço

```
(define-struct bqd (nome preço))
;; Um bqd é uma estrutura
;;  (make-bqd n p),
;; onde n é um símbolo e p é um número
```

Lista de Estruturas (cont.)

- O inventário de uma loja pode ser:
  - Vazio (`empty`) ou
  - `(cons bqd invent)`, onde *bqd* é um brinquedo e *invent* é um inventário

- Exemplos de inventário:

```
(cons (make-bqd 'boneca 17.95) empty)

(cons (make-bqd 'robô 22.05)
   (cons (make-bqd 'boneca 17.95)
      empty))
```

Lista de Estruturas (cont.)

- **Exemplo:** Função que calcula a soma total dos preços dos brinquedos de uma loja.

```
;; soma: inventário -> number
;; Calcula a soma total dos preços dos brinquedos
;; do inventário de uma loja

(define (soma inv) ...)
```

Lista de Estruturas (cont.)

- Inicialmente, definem-se as cláusulas da função.

```
(define (soma inv)
   (cond
      [(empty? inv)...]
      [else ... (first inv)...(soma(rest inv))...]))
```

Lista de Estruturas (cont.)

```
(define (soma inv)
   (cond
      [(empty? inv) 0]
      [else (+ (bqd-price (first inv))(soma (rest inv)))]))
```

- `(first inv)` extrai o primeiro elemento da lista
- `(bqd-price (first inv))` obtém o preço do brinquedo localizado no ínicio da lista
- `(soma (rest inv))` obtém o resto da lista e calcula a soma com uma chamada recursiva

Lista de Estruturas (cont.)

- Suponha que a loja decida vender os brinquedos que custam abaixo de $1 em um outro departamento

- **Problema:** Desenvolver uma função que, dada a lista de brinquedos de uma loja, constrói uma lista dos brinquedos que custam menos que $1

```
;; cria-lista: inventário -> inventário
;; Dada a lista de brinquedos de uma loja, cria uma segunda
;; lista apenas com os brinquedos que custam menos do que $1

(define (cria-lista inv)...)
```

Lista de Estruturas (cont.)

- Exemplo:

```
(cons (make-bqd 'espada 0.95)
   (cons (make-bqd 'Barbie 17.95)
      (cons (make-bqd 'chaveiro 0.55)
         (cons (make-bqd 'robô 22.95)
            empty))))
```

- produz:

```
(cons (make-bqd 'espada .95)
   (cons (make-bqd 'chaveiro .55)
      empty))
```

Lista de Estruturas (cont.)

```
(define (cria-lista inv)
   (cond
      [(empty? inv)...]
      [else ... (first inv)...(cria-lista (rest inv))...]))
```

Lista de Estruturas (cont.)

```
;; cria-lista: inventário -> inventário
;; Dada a lista de brinquedos de uma loja, cria uma segunda
;; lista apenas com os brinquedos que custam menos que $1

(define (cria-lista inv)
   (cond
      [(empty? inv) empty]
      [else
        (cond
          [(<= (bqd-price (first inv)) 1.00)
            (cons (first inv)(cria-lista (rest inv)))]
          [else (cria-lista (rest inv))])]))
```

Exercício

- Desenvolva uma função `aumenta-preços` que recebe uma lista de brinquedos e retorna esta lista com os preços dos brinquedos acrescidos de 5% do seus valores originais.