Fundamentos de Algoritmos

CIC/UFRGS

2006/01

Listas de Estruturas

- vimos listas com dados atômicos
- as listas também podem conter dados compostos
- funções podem produzir listas
- além disso, funções podem receber e 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 por cada funcionário, produzir uma lista dos salários correspondentes, calculados com base no valor de \$12 por hora.

```
;; salário: number -> number
;; calcula o salário calculado com base em \ $12 por hora
(define (salário h)
  (* 12 h))
```

Função que cria uma lista

```
;; horas->salário: lista-de-números -> lista-de-números
;; cria uma lista de salários dada uma lista das horas trabalhadas
(define (horas->salário alon)...
```

- Exemplos de entradas -> saídas correspondentes

<code>empty</code>	<code>empty</code>
<code>(cons 28 empty)</code>	<code>(cons 336 empty)</code>
<code>(cons 40 (cons 28 empty))</code>	<code>(cons 480 (cons 336 empty))</code>

Função que cria uma lista

- como toda função recursiva, pelo menos duas cláusulas aparecem no corpo da função

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

Função que cria uma lista

```
;; horas->salário: lista-de-números -> lista-de-números
;; cria uma lista de salários dada uma lista das horas trabalhadas
(define (horas->salário alon)
  (cond
    [(empty? alon) empty]
    [else (cons(salário(first alon))(horas->salário(rest alon)))]))

;; salário: number -> number
;; calcula o salário considerando o pagamento de $12 por hora
(define (salário h)
  (* 12 h))
```

Exercício

1. Desenvolva uma função `elimina-caros` que elimina brinquedos caros. A função recebe um número (*max*), e uma lista de preços de brinquedos `/preço`, e produz uma lista daqueles brinquedos `/preço` com preço igual ou menor a *max*. Por exemplo:

```
(elimina-caros 1.0 (cons 2.95 (cons 0.95 (cons 1.0 (cons 5 empty))))))  
;expected value: (cons 0.95 (cons 1.0 empty))
```

Lista que contém estruturas

- uma lista não necessariamente precisa conter dados atômicos
- abordaremos aqui o caso de 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: nome e preço

```
(define-struct bqd (nome preço))
```

Um brinquedo é uma estrutura (make-bqd n p)
onde n é um símbolo e p é um número

Lista de estruturas

- o inventário de uma loja pode ser:
 - vazio 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 'robo 22.05)  
      cons (make-bqd 'boneca 17.95)  
          empty))
```

Lista de estruturas

- **Exemplo:** desenvolva uma 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 uma loja
(define (soma inv) ...)
```

Lista de estruturas

- inicialmente define-se as cláusulas da função.

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

Lista de estruturas

```
(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 início da lista
- (soma (rest inv)) obtém o resto da lista e calcula a soma com uma chamada recursiva

Lista de estruturas

- Suponha que a loja decide vender os brinquedos que custam abaixo de \$1 em um outro departamento.
- **Problema:** desenvolva 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 com apenas os brinquedos que custam menos que $1
(define (cria-lista inv)...)

```

Lista de estruturas

- Exemplo:

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

- produz:

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

Lista de estruturas

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

Lista de estruturas

```
;;cria-lista: inventário -> inventário
;;dada a lista de brinquedos de uma loja, cria uma segunda
;;lista com apenas 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 `raise-prices` o qual recebe uma lista de brinquedos e retorna esta lista com o preço dos brinquedos acrescido em 5%.