Fundamentos de Algoritmos

CIC/UFRGS

2006/1

1

Funções Compostas

- Um programa é composto por definições de funções e variáveis
- Funções auxiliares são definidas para processar dependências
- Nesta seção funções auxiliares serão propostas com um objetivo mais amplo

Roteiro

- Projeto de programas complexos
- Funções recursivas auxiliares
- Exemplos
- Exercícios

2

Projeto de programas complexos

- Somente programas muito simples são compostos por apenas uma função
- Tratar as dependências usando funções auxiliares facilita o entendimento do programa
- Antes de iniciar um programa, os dados de entrada e saída devem ser analisados

3

Projeto de programas complexos

- Situações diferentes são encontradas:
 - se a formulação de uma resposta depende de uma avaliação de números, usa-se cond
 - se a função requer o processamento de dados de um domínio particular, usam-se funções auxiliares
 - caso deve-se processar um número natural, uma lista ou outra forma de dados de tamanho arbitrário, usam-se funções auxiliares
 - se a função principal apenas enumera os processamentos que devem ser feitos, esta é composta por chamadas a funções auxiliares
- Neste capítulo daremos ênfase aos dois últimos itens

Projeto de Programas complexos

- Ao determinar que funções auxiliares são necessárias:
 - um contrato, objetivo e cabeçalho são desenvolvidos para cada uma
 - Desenvolva cada uma de acordo com seu objetivo
- se a função já existe ou se já está na lista, não redefini-la
- testar cada função individualmente
- testa o programa com todas as funções

Funções auxiliares recursivas

- ordenação é uma função que comumente aparece em programas
- para tanto uma função auxiliar é usada para fazer ordenação de dados
- números podem ser fornecidos por uma lista
- a saída é uma lista de números com os dados de entrada ordenados

Ordenação de números

```
;; ordena: lista-de-números -> lista-de-números
;; gera uma lista ordenada dos números fornecidos numa
;; lista de entrada
(define (ordena l-num)...

;; Exemplos:
;; (ordena empty) produz empty
;; (ordena (cons 12 (cons 20 (cons -5 empty))))
;;           produz (cons 20 (cons 12 (cons -5 empty)))
```

Ordenação de números

```
(define (ordena l-num)
  (cond
    [(empty? l-num) ...]
    [else...(first l-num)...(ordena (rest l-num))...]))
```

1. (first l-num) extrai o primeiro número da lista
2. (ordena (rest l-num)) produz uma lista ordenada dos números da lista (rest l-num)

Ordenação de números

- (first l-num) produz 12
- (rest l-num) é (cons 20 (cons -5 empty))
- (ordena (rest l-num)) produz (cons 20 (cons -5 empty))
- deve-se inserir 12 entre 20 e -5
- usa-se uma expressão cond que insere (first l-num) na posição correta de (ordena (rest l-num))
- uma função auxiliar pode ser criada com esse intuito

Ordenação de números

```
;; insere: número lista-de-números -> lista-de-números
;; gera uma lista ordenada de números a partir de n
;; e uma lista já ordenada
(define (insere n l-num)...)
```

Ordenação de números

Usando insere pode-se completar a definição de ordena

```
;; ordena: lista-de-números -> lista-de-números
;; gera uma lista ordenada dos números fornecidos pela
;; lista de entrada
(define (ordena l-num)
  (cond
    [(empty? l-num) empty]
    [else insere(first l-num)(ordena (rest l-num))]))

;;(insere -5 empty) produz (cons -5 empty)
;;(insere 12 (cons 20 (cons -5 empty)))
;; produz (cons 20 (cons 12 (cons -5 empty)))
```

Ordenação

```
(define (insere n l-num)
  (cond
    [(empty? l-num)...]
    [else...(first l-num)...(insere n (rest l-num))...]))
```

Ordenação

- para (insere 7 (cons 6 (cons 5 (cons 4 empty)))) o novo elemento é apenas inserido normalmente na lista
- para (insere 3 (cons 6 (cons 2 (cons 1 (cons -1 empty))))), o n deve ser inserido entre números da lista
 - (first l-num) é 6
 - (insere n (rest l-num)) é (cons 3 (cons 2 (cons 1 (cons -1 empty))))
- E se n é menor que o primeiro número da lista??

Ordenação

```
(cond
  [(>= n (first l-num)) ...]
  [(< n (first l-num)) ...])
```

Ordenação de lista de num

```
;; ordena: lista-de-números -> lista-de-números
;; gera uma lista ordenada dos números informados na lista de entrada
(define (ordena l-num)
  (cond
    [(empty? l-num) empty]
    [(cons? l-num)(insere(first l-num)(ordena (rest l-num)))]))

;; insere: número lista-de-números -> lista-de-números
;; gera uma lista ordenada de núm. a partir de n e uma lista ordenada
(define (insere n l-num)
  (cond
    [(empty? l-num) (cons n empty)]
    [else (cond
      [{>= n (first l-num)} (cons n l-num)]
      [(< n (first l-num))(cons (first l-num)(insere n (rest l-num)))]))]))
```

Exercícios

- Desenvolva uma função que ordena uma lista de e-mails pela data. A estrutura de um e-mail é definida da seguinte forma: (define-struct mail (from data message)) onde *nome* e *s* são strings e *n* é um número.
- Desenvolva uma função que ordena uma lista de e-mails pelo nome. Para comparar dois strings alfabéticos, use a primitiva `string<?`.

Exercícios

```
;; busca: número lista-de-números -> boolean
(define (busca n l-num)
  (cond
    [(empty? l-num) false]
    [else (or (= (first l-num)n)(busca n (rest l-num)))]))
```

A função `busca` acima determina se um número ocorre numa lista de números. Todos os números podem ser consultados na busca de um em específico. Desenvolva a função `busca-ordenada` o qual determina se um número ocorre em uma lista ordenada de números. A função deve aproveitar o fato de que a lista é ordenada. Esse tipo de busca é chamado de `busca-linear`.

Generalização de Problemas e Funções

- considere o problema de desenhar um polígono
- um polígono é uma figura geométrica formada por um número arbitrário de ângulos
- pode ser representado por uma lista de estruturas `posn`

Desenhando Polígonos

- uma lista-de-posn é
 - vazia ou
 - (cons p lp) onde *p* é uma estrutura `posn` e *lp* é uma lista de `posn`

Desenhando Polígonos

- cada posn representa um ângulo do polígono

```
(cons (make-posn 10 10)
      (cons (make-posn 60 60)
            (cons (make-posn 10 60)
                  empty))))
```

Desenhando Polígonos

Um polígono é

- (cons p empty) onde p é uma estrutura posn
- (cons p lp) onde p é uma estrutura posn e lp é um polígono

Desenhando Polígonos

```
;;desenha-poli: polígono -> true
;;desenha um polígono especificado por poli
(define (desenha-poli poli)
  (cond
    [(empty? (rest poli))...(first poli)...]
    [else ...(first poli)...
           ...(second poli)...
           (desenha-poli (rest poli))...]))
```

Desenhando Polígonos

```
;;desenha-poli: polígono -> true
;;desenha um polígono especificado por poli
(define (desenha-poli poli)
  (cond
    [(empty? (rest poli)) true]
    [else (and (draw-solid-line (first poli)(second poli))
               (desenha-poli (rest poli)))]))
```

Desenhando Polígonos

- para adicionar a última aresta pode-se
 - adicionar a primeira posn estrutura também no final da lista
 - adicionar a última posn também no início da lista
 - modificar esta função de tal forma que uma linha também é desenhada conectando a primeira e a última estruturas posn
- vamos desenvolver a segunda opção e os exercícios tratam dos outros dois casos

Desenhando Polígonos

```
;; último: polígono -> posn
;; extrai o último posn de um polígono
(define (último poli)...
```

Desenhando Polígonos

```
;; último: polígono -> posn
;; extrai o último posn de um polígono
(define (último poli)
  (cond
    [(empty? (rest poli))...(first poli)...]
    [else ... (first poli)...
      ... (second poli)...
      (último (rest poli))...]))
```

Desenhando Polígonos

```
;; último: polígono -> posn
;; extrai o último posn de um polígono
(define (último poli)
  (cond
    [(empty? (rest poli))(first poli)]
    [else (último (rest poli))]))
```

Desenhando Polígonos

```
;;desenha-poli: polígono -> true
;;desenha um polígono especificado por poli
(define (desenha-poli poli)
  (conecta-pontos (cons (último poli) poli)))

;;conecta-pontos: polígono -> true
;;desenha as arestas entre ângulos de um polígono poli
(define (conecta-pontos poli)
  (cond
    [(empty? (rest poli)) true]
    [else (and (draw-solid-line (first poli)(second poli) RED)
                (conecta-pontos (rest poli)))]))

;; último: polígono -> posn
;; extrai o último posn de um polígono
(define (último poli)
  (cond
    [(empty? (rest poli))(first poli)]
    [else (último (rest poli))]))
```