Fundamentos de Algoritmos - Expressões Condicionais

INF05008

(Cap. 4)

Expressões e Funções Condicionais

Para diversos problemas, programa deve lidar com situações diferentes de formas diferentes

- jogo deve determinar se velocidade está dentro de um intervalo
- ou se um objeto está em determinada posição do vídeo
- frações, ou racionais
- em controle de processos condição determina se válvula deve ou não ser aberta
- ...

Booleanos e Relações

- condições são comuns em matemática
- exemplos: um número pode ser igual, menor ou maior a outro número
- se x e y são números :
 - $x = y$
 - $x < y$ e
 - $x > y$

- para cada par de números somente uma das afirmações acima é verdadeira
- se $x = 4$ e $y = 5$ somente a segunda afirmação é verdadeira e as outras são falsas
- se $x = 5$ e $y = 4$ a terceira afirmação é verdadeira e as outras são falsas
- em geral uma afirmação é verdadeira para alguns valores e falsa para outras

- afirmações atômicas podem ser combinadas em afirmações compostas
- exemplos de combinações das afirmações atômicas acima:
 1. $x = y$ e $x < y$ e $x > y$
 2. $x = y$ ou $x < y$ ou $x > y$
 3. $x = y$ ou $x < y$

- $x = y$ e $x < y$ e $x > y$ é sempre falsa
- $x = y$ ou $x < y$ ou $x > y$ é sempre verdadeira
- $x = y$ ou $x < y$ verdadeira em alguns caso e falsa em outros casos
- exemplos: verdadeira quando $x = 4, y = 4$, verdadeira quando $x = 4, y = 5$, falsa quando $x = 5, y = 3$

- em Scheme:
 - true
 - false
 - (= x y) *x é igual a y*
 - (< x y) *x é menor do que y*
 - (> x y) *x é maior do que y*

- Exemplos de condições em Scheme

- `( < 4 5 )`
- `( and (= x y) (< y z) )`
- `( or (= x y) (< y z) )`
- `( and ( = 5 5) (< 5 6) )`

Exercícios

Exercício 4.1.1. Qual o resultado das seguintes condições em Scheme?

1. `(and (> 4 3) (<= 10 100))`
2. `(or (> 4 3) (<= 10 100))`
3. `(not (= 2 3))`

Exercício 4.1.2. Qual o resultado de

1. `(> 4 3)`
2. `(and (> 4 x) (> x 3))`
3. `(= (* x x) x)`

para (a) $x = 4$, (b) $x = 2$, (c) $x = \frac{7}{2}$

Funções que Testam Condições

Exemplo de função que testa uma condição sobre um número:

```
;; is-5? : numero --> boolean  
;; determina se n igual a 5
```

```
(define (is-5? n)  
  (= n 5))
```

A função produz `true` se e somente se sua entrada é igual a 5

Outros exemplos de funções que testam condições:

```
;; is-between-5-6? : numero --> boolean  
;; determina se n esta entre 5 e 6 (exclusivo)
```

```
(define (is-between-5-6? n)  
  (and (< 5 n) (< n 6)))
```

```
;; is-between-5-6-or-over-10? : numero --> boolean  
;; determina se n esta entre 5 e 6 (exclusivo)  
;; ou e maior do que 10
```

```
(define (is-between-5-6-or-over-10? n)
```

```
(or (is-between-5-6? n) (>= n 10))
```

Exercícios

Exercícios 4.2.1. Traduza os intervalos abaixo para funções em Scheme que aceitam um número e produzem `true`, se o número está no intervalo, e `false` se está fora do intervalo:

1. $(3, 7]$
2. $[3, 7]$
3. $[3, 9)$
4. união de $(1, 3)$ e $(9, 11)$
5. na parte fora de $[1, 3]$

Exercícios

Exercícios 4.2.2. Traduza as funções abaixo para intervalos

```
(define (in-interval-1? x)
  (and (< -3 x) (< x 0)))
```

```
(define (in-interval-2? x)
  (or (< x 1) (> x 2)))
```

```
(define (in-interval-3? x)
  (not (and (<= 1 x) (<= x 5))))
```

Escreva os contratos e propósitos para cada uma das funções acima.

Condicionais e Funções Condicionais

Formato geral de EXPRESSÕES CONDICIONAIS:

```
(cond  
  [question answer]  
  ...  
  [question answer])
```

OU

```
cond  
  [question answer]  
  ...  
  [else answer])
```

Exemplo:

```
(cond
  [(< n 10) 5.0]
  [(< n 20) 5]
  [(< n 30) true])
```

Exemplo de expressão condicional mal formada

```
(cond
  [(< n 10) 30 12]
  [(> n 25) false]
  [(> n 20) 0])
```

- Scheme determina o valor de cada condição
- uma condição avalia para `true` ou `false`
- para a primeira que avaliar `p/true`, Scheme avalia a *answer* correspondente
- o valor desta resposta é o valor final da expressão condicional
- se a última condição é um `else`, e todas as demais falham, a última resposta é o valor da expressão condicional

Eis dois exemplos

```
(cond
  [(<= n 1000) .040]
  [(<= n 5000) .045]
  [(<= n 10000) .055]
  [(> n 10000) .060])
```

e

```
(cond
  [(<= n 1000) .040]
  [(<= n 5000) .045]
  [(<= n 10000) .055]
  [else .060])
```

Avalie as expressões para $n = 10000$, $n = 20000$

Exercício 4.3.1. Qual das duas expressões abaixo é legal?

```
(cond
  [(< n 10) 20]
  [(> n 20) 0]
  [else 1])
```

```
(cond
  [(< n 10) 20]
  [(and (> n 20) (<= n 30))]
  [else 1])
```

Exercício 4.3.1 (cont.) Por que a seguintes expressão condicional é ilegal?

```
(cond
  [(< n 10) 20]
  [* 10 n]
  [else 555])
```

Suponha que um banco pague juros de 4% para depósitos de até 1000 (inclusive), 4.5% para depósitos de até 5000 (inclusive) de 5% para depósitos de mais de 5000. Escreva um programa que, dado o valor a ser depositado, produza a taxa de juro correspondente a esse valor.

Contrato

```
;; interest-rate : number -> number
;; determinar a taxa de juros, dada uma quantia

(define (interest-rate amount)
  ...)
```

Eis alguns exemplos de uso:

```
(= (interest-rate 1000) .040)
```

```
(= (interest-rate 5000) .045)
```

```
(= (interest-rate 8000) .050)
```

O corpo da função deve ser uma expressão condicional que distingue os três casos mencionados no enunciado do problema.

```
(cond
  [(<= amount 1000) ...]
  [(<= amount 5000) ...]
  [> amount 5000) ...])
```

Usando os exemplos e o *rascunho* da expressão condicional a resposta é fácil:

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))
```

Como sabemos que a função só precisa de três casos, podemos trocar a última condição por um `else`:

```
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [else 0.050]))
```

Quando aplicamos *interest-rate* para uma quantia, 4000, por exemplo, o cálculo segue como esperado: Scheme primeiro copia o corpo da função e troca *amount* por 4000

```
(interest-rate 4000)
= (cond
  [(<= 4000 1000) 0.040]
  [(<= 4000 5000) 0.045]
  [else 0.050])
= 0.045
```

Resultado Final do Processo

```
;; interest-rate : number -> number
;; determina a taxa de juros, dada uma quantia
;; exemplos de uso:
;;   (= (interest-rate 1000) .040)
;;   (= (interest-rate 5000) .045)
;;   (= (interest-rate 8000) .050)

(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [else 0.050]))
```

Projetando Funções Condicionais - Etapas

Fase: Análise de dados

Objetivo: determinar as situações distintas com as quais a função deve lidar

Atividade: inspecionar o enunciado do problema para identificar situações distintas; listar essas situações

Fase: Exemplos

Objetivo: fornecer um exemplo para cada situação

Atividade: escolher pelo menos um exemplo para cada situação para intervalo ou enumerações. Os exemplos devem contemplar casos em limites

Fase: Corpo da função (1)

Objetivo: formular a expressão condicional

Atividade: escrever o esqueleto para a expressão condicional, com uma cláusula por situação

Fase: Corpo da função (2)

Objetivo: formular as respostas para a expressão condicional

Atividade: lidar com cada linha da expressão condicional em separado e desenvolver a expressão Scheme que computa a resposta apropriada para cada caso

Exercícios

Exercício 4.4.1. Desenvolva a função *interest* que consome uma quantia depositada e produz o rendimento recebido em um ano por essa quantia. O banco paga 4% para depósitos até 1000; 4.5% para depósitos até 5000, e 5% para depósitos acima de 5000.

Exercício 4.4.2. Desenvolva a função *tax*, que consome um salário bruto e produz a quantia de imposto devido. Para salário bruto de 240 ou menos, o imposto é de 0%; acima de 240 até 480 o imposto é de 15%. Acima de 480 o imposto é de 28%.

Desenvolva também a função *net* que calcula o salário líquido de um trabalhador a partir do número de horas trabalhadas. O salário líquido é o bruto menos o imposto. Assuma que o valor da hora de trabalho é de \$12.