Expressões Condicionais

Fundamentos de Algoritmos

INF05008

Expressões e Funções Condicionais

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

- Jogo deve determinar se a 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 do que outro número
- Se x e y são números:
  - $x = y$,
  - $x < y$ ou
  - $x > y$

- Para cada par x, y de números, somente uma das afirmações acima é verdadeira
- Se $x = 4$ e $y = 5$, 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 outros

- Afirmações **atômicas** podem ser combinadas em afirmações **compostas**

- Exemplos de combinações das afirmações atômicas anteriores:

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 casos e **falsa** em outros casos
- Exemplos: verdadeira quando $x = 4$ e $y = 4$, verdadeira quando $x = 4$ e $y = 5$, falsa quando $x = 5$ e $y = 3$, ...

- Em Scheme:
  - `true` *verdadeiro*
  - `false` *falso*
  - `(= 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))`

Funções que Testam Condições

Exemplo de função que testa uma condição sobre um número:

```
;; é-5? : número -> boolean
;; Determina se 'n' igual a 5

(define (é-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:

```
;; entre-5-6? : número -> boolean
;; Determina se 'n' está entre 5 e 6 (exclusivo)

(define (entre-5-6? n)
    (and (< 5 n) (< n 6)))

;; entre-5-6-ou-acima-10? : número -> boolean
;; Determina se 'n' está entre 5 e 6 (exclusivo)
;; ou é maior do que 10

(define (entre-5-6-ou-acima-10? n)
    (or (entre-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`, caso contrário.

1. $(3, 7]$
2. $[3, 7]$
3. $[3, 9)$
4. União de $(1, 6)$ e $(9, 14)$
5. Na parte fora de $[1, 3]$

Exercícios (cont.)

Exercícios 4.2.2. Traduza as funções abaixo para intervalos:

```
(define (no-intervalo-1? x)
    (and (< -3 x) (< x 0)))

(define (no-intervalo-2? x)
    (or (< x 1) (> x 2)))

(define (no-intervalo-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
 [pergunta resposta]
 ...
 [pergunta resposta])
```

ou

```
(cond
 [pergunta resposta]
 ...
 [else resposta])
```

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 para `true`**, Scheme avalia a *resposta* 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 $n = 10000$ e $n = 20000$.

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])
```

Por quê a seguinte expressão condicional é **ilegal**?

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

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

Contrato

```
;; taxa-de-juros : número  ->  número
;; Determina a taxa de juros, dada uma quantia

(define (taxa-de-juros quantia)
          ...)
```

Eis alguns exemplos de uso:

```
(= (taxa-de-juros 1000) .040)

(= (taxa-de-juros 5000) .045)

(= (taxa-de-juros 8000) .050)
```

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

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

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

```
(define (taxa-de-juros quantia)
  (cond
    [(<= quantia 1000) 0.040]
    [(<= quantia 5000) 0.045]
    [(> quantia 5000) 0.050]))
```

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

```
(define (taxa-de-juros quantia)
  (cond
    [(<= quantia 1000) 0.040]
    [(<= quantia 5000) 0.045]
    [else 0.050]))
```

Quando aplicamos *taxa-de-juros* para uma quantia - $R\$4000$, por exemplo -, o cálculo segue como esperado: Scheme primeiro copia o corpo da função e, depois, troca *quantia* por $R\$4000$:

```
   (taxa-de-juros 4000)
= (cond
    [(<= 4000 1000) 0.040]
    [(<= 4000 5000) 0.045]
    [else 0.050])
= 0.045
```

Resultado Final do Processo

```
;; taxa-de-juros : número  ->  número
;; Determina a taxa de juros, dada uma quantia
;; Exemplos de uso:
;;    (= (taxa-de-juros 1000) .040)
;;    (= (taxa-de-juros 5000) .045)
;;    (= (taxa-de-juros 8000) .050)

(define (taxa-de-juros quantia)
  (cond
    [(<= quantia 1000) 0.040]
    [(<= quantia 5000) 0.045]
    [else 0.050]))
```

Projetando Funções Condicionais - Etapas

Fase 1: 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 estas situações

Fase 2: Exemplos

Objetivo: Fornecer um exemplo **para cada situação**

Atividade: Escolher pelo menos um exemplo para cada situação para intervalos ou enumerações. Os exemplos devem contemplar casos em **limites**

Fase 3: Corpo da função (1)

Objetivo: Formular a **expressão condicional**

Atividade: Escrever o **esqueleto** da expressão condicional, com **uma cláusula por situação**

Fase 4: 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