Tipos Mistos

Fundamentos de Algoritmos

INF05008

Vários Tipos de Dados

Até aqui, as funções que definimos usavam 4 tipos de dados:

- `number`: representando **informações numéricas**;
- `boolean`: representando **valores-verdade**;
- `symbol`: representando **informação simbólica**;
- `struct`: representando **composição de informações**.

Às vezes, precisamos definir funções que processam uma **classe de dados** , incluindo números e estruturas ou estruturas de vários tipos diferentes, por exemplo. Como podemos **definir** essas funções e também **proteger** nossas funções de usos indevidos?

Misturando e Distinguindo Dados

Problema: Uma definição alternativa de posições em um plano usa, em vez de um par de coordenadas (x, y), apenas a coordenada x caso o ponto esteja no eixo x (ou seja, quando $y = 0$). Construa uma função que calcula a distância de um ponto até o 0, levando em consideração que a entrada pode ser do tipo (x, y) ou do tipo x.

Como descobrir o **tipo da entrada** , já que esta pode ser uma estrutura (`posn`) ou um número (`number`)?

Predicados para Distinguir Dados

Scheme oferece **predicados** predefinidos para identificar tipos de dados:

- `number?`: Retorna `true` se o valor ao qual a função é aplicada é um **número** e `falso`, caso contrário;
- `boolean?`: Retorna `true` se o valor ao qual a função é aplicada é um **valor-verdade** e `falso`, caso contrário;
- `symbol?`: Retorna `true` se o valor ao qual a função é aplicada é um **símbolo** e `falso`, caso contrário;
- `struct?`: Retorna `true` se o valor ao qual a função é aplicada é uma **estrutura** e `falso`, caso contrário.

Predicados para Distinguir Dados

Além disso, para cada definição de estrutura são gerados, **automaticamente**, predicados para identificar valores dessas novas classes. Por exemplo, para as definições a seguir

```
(define-struct posn (x y))

(define-struct estrela (sobrenome nome instrumento vendas))

(define-struct avião (tipo veloc-máx capacidade preço))
```

são gerados os seguintes predicados:

- `posn?`
- `estrela?`
- `avião?`

Exercícios

Avalie as seguintes expressões

1. `(number? (make-posn 2 3))`

2. `(number? (+ 12 10))`

3. `(posn? 23)`

4. `(posn? (make-posn 23 3))`

5. `(estrela? (make-posn 23 3))`

Função `distância-para-0`

Definição de dados: Um `pixel-2` é

1. ou um número (`number`),

2. ou uma estrutura `posn`.

Contrato, objetivo e cabeçalho:

```
;; distância-para-0 : pixel-2  ->  número

;; Calcular a distância de um ponto (um-pixel)
;; para a origem

(define (distância-para-0 um-pixel) ...)
```

Função `distância-para-0`

Vamos definir a função por etapas...

Etapa 1: Como identificar o **tipo da entrada** ?

Função `distância-para-0`

Etapa 1: Como identificar o **tipo da entrada** ?

Usando uma expressão `cond` :

```
(define (distância-para-0 um-pixel)
  (cond
    [(number? um-pixel) ...]
    [(posn? um-pixel) ...]))
```

Função `distância-para-0`

Etapa 2: Como **selecionar as coordenadas** , caso a entrada seja uma estrutura `posn`?

Função `distância-para-0`

Etapa 2: Como **selecionar as coordenadas** , caso a entrada seja uma estrutura `posn`?

Usando as funções auxiliares `posn-x` e `posn-y` :

```
(define (distância-para-0 um-pixel)
  (cond
    [(number? um-pixel) ...]
    [(posn? um-pixel)...(posn-x um-pixel)...
                        (posn-y um-pixel)...]))
```

Função `distância-para-0`

Etapa 3: Como **calcular a função** em cada caso?

Função `distância-para-0`

Etapa 3: Como **calcular a função** em cada caso?

```
(define (distância-para-0 um-pixel)
  (cond
    [(number? um-pixel) um-pixel]
    [(posn? um-pixel)
      (sqrt (+ (sqr (posn-x um-pixel))
               (sqr (posn-y um-pixel))))]))
```

Desenvolvendo Funções com Dados Mistos

Fases do projeto de algoritmos a serem modificadas:

- **Análise e projeto de dados**: Determinar classes distintas de dados . Definição de dados terá cláusulas enumerando os tipos de dados do problema;

- **Construção do *template***: Corpo da função é uma expressão `cond` com tantas cláusulas quantos são os tipos de dados envolvidos no problema;

- **Definição da função**: Usando o condicional, o problema é dividido em vários subproblemas. Cada cláusula de `cond` é tratada separadamente.

As outras fases (**Contrato**, **Objetivo**, **Cabeçalho**, **Exemplos** e **Testes**) não sofrem modificações pelo uso de dados mistos.

Projeto de Algoritmos usando Dados Mistos

Fase	**Objetivo**	**Atividade**
Projeto e Análise de Dados	Formular uma definição de dados	Determinar quantas classes distintas de objetos tem o problema, enumerar as alternativas em uma definição de dados, fazer as definições de estruturas que forem necessárias
Contrato, Objetivo e Cabeçalho	Dar um nome à função, especificar as classes de entrada e saída, descrever o objetivo e formular um cabeçalho	Nomear a função, as classes de entrada e saída e especificar um objetivo

Fase	**Objetivo**	**Atividade**
Exemplos	Caracterizar a relação entrada-saída através de exemplos	Criar exemplos da relação entrada-saída, levando em consideração que deve existir pelo menos um exemplo para cada subclasse de dados a qual a função pode ser aplicada
Template	Formular um esboço para a função	Criar uma estrutura `cond` com uma linha para cada tipo de dado. Identificar cada um dos tipos usando predicados de identificação de tipos

Fase	**Objetivo**	**Atividade**
Corpo	Completar a definição da função	Construir expressões Scheme para cada uma das cláusulas do `cond`
Testes	Encontrar erros	Aplicar a função aos exemplos e verificar se os resultados são os esperados

Função que calcula o perímetro de uma forma

```
;; Definição de dados:

(define-struct círculo (centro raio))
(define-struct quadrado (nw lado))

;; Uma forma é sempre 1) uma estrutura
;; (make-círculo c r),
;; onde 'c' é do tipo posn e 'r' é um número;
;; ou 2) uma estrutura
;; (make-quadrado n l),
;; onde 'n' é do tipo posn e 'l' é um número.
```

Função que calcula o perímetro de uma forma (cont.)

```
;; Contrato, Objetivo, Cabeçalho:

;; perímetro : forma  ->  número
;; Computar o perímetro de uma forma

(define (perímetro uma-forma)
  ... )

;; Exemplos: considere os testes
```

Função que calcula o perímetro de uma forma (cont.)

```
;; Template:

;; (define (f uma-forma)
;;   (cond
;;     [(quadrado? uma-forma)
;;        ...(quadrado-nw uma-forma)...(quadrado-comprimento
;;        uma-forma)...]
;;     [(círculo? uma-forma)
;;        ...(círculo-centro uma-forma)...(círculo-raio
;;        uma-forma)...]))
```

Função que calcula o perímetro de uma forma (cont.)

```
;; Definição da função:

(define (perímetro uma-forma)
  (cond
    [(círculo? uma-forma)
     (* (* 2 (círculo-raio uma-forma)) PI)]
    [(quadrado? uma-forma)
     (* (quadrado-lado uma-forma) 4)]))
```

Função que calcula o perímetro de uma forma (cont.)

```
;; Testes:

(= (perímetro (make-quadrado ... 3)) 12)

(= (perímetro (make-círculo ... 1)) (* 2 PI))
```

Exercício

Desenvolva estruturas e definições de dados para o tipo de dados `forma`, o qual representa objetos tridimensionais. A coleção deve incluir:

- Cubos: o atributo relevante é o tamanho do lado;
- Prismas: sólidos retangulares cujos atributos relevantes são a altura, a largura e a profundidade;
- Esferas: o atributo importante é o raio.

1. Desenvolva a função `volume`, que consome uma forma tridimensional e produz o volume da forma. Dica : o volume de uma esfera de raio r é $4/3 * \pi * r^3$.

2. Desenvolva a função `mesma-forma?`, que consome duas formas tridimensionais e retorna `true` somente se ambas possuírem **exatamente** as mesmas dimensões.

Soluções

```
(define PI 3.14)

(define-struct cubo (lado))

;; Um cubo é uma estrutura
;; (make-cubo lado)
;; onde 'lado' é um número

(define-struct prisma (altura largura profundidade))

;; Um prisma é uma estrutura
;; (make-prisma altura largura profundidade)
;; onde 'altura', 'largura' e 'profundidade' são números
```

Soluções (cont.)

```
(define-struct esfera (raio))

;; Uma esfera é uma estrutura
;; (make-esfera raio)
;; onde 'raio' é um número

;; Uma forma pode ser
;; 1) uma estrutura cubo
;; 2) uma estrutura prisma, ou
;; 3) uma estrutura esfera
```

Soluções (cont.)

```
;; volume : forma -> número
;; Calcula o volume de uma forma tridimensional.
;; Assume-se que a base de um prisma é um retângulo

(define (volume uma-forma)
  (cond
    [(cubo? uma-forma) (expt (cubo-lado uma-forma) 3)]
    [(prisma? uma-forma) (* (* (prisma-largura uma-forma)
                               (prisma-profundidade uma-forma))
                            (prisma-altura uma-forma))]
    [(esfera? uma-forma) (* (* (/ 4 3) PI)
                            (expt (esfera-raio uma-forma) 3))]))
```

Soluções (cont.)

```
;; mesma-forma? : forma1 forma2 -> boolean
;; Identifica formas iguais que possuem as
;; mesmas dimensões

(define (mesma-forma? forma1 forma2)
  (cond
    [(and (cubo? forma1) (cubo? forma2))
     (= (cubo-lado forma1) (cubo-lado forma2))]
    [(and (prisma? forma1) (prisma? forma2))
     (and (= (prisma-largura forma1)
             (prisma-largura forma2))
          (= (prisma-profundidade forma1)
             (prisma-profundidade forma2))
          (= (prisma-altura forma1)
             (prisma-altura forma2)))]
    [(and (esfera? forma1) (esfera? forma2))
     (= (esfera-raio forma1) (esfera-raio forma2))]
    [else false]))
```