Dados Compostos: Estruturas

- A entrada (elemento do domínio) para uma função em geral é um dado que representa um objeto e suas propriedades.
- Exemplo: entrada pode ser informação sobre um CD (artista, gravadora, nome, preço);
- Vários dados são COMPOSTOS em um único dado.
- Scheme oferece várias formas de compor dados,
- entre elas: *estruturas*

- Estrutura para representar um *pixel*
- Um pixel possui uma coordenada x e outra y
- Os *teachpacks* do DrScheme representam um pixel com uma estrutura *posn*

- Um único dado *posn* contém dois valores.
- Uma estrutura *posn* é criada com a operação *make-posn* que consome dois números e produz um *posn*
- Exemplos:

```
(make-posn 3 4)
```

```
(make-posn 8 6)
```

```
(make-posn 5 12)
```

Considere uma função que computa a distância de um pixel da origem:

```
;; distance-to-0 : posn -> number
;; objetivo: calcular a distância do ponto a-posn
;;           até a origem
```

```
(define (distance-to-0 a-posn) ...)
```

A função `distance-to-0` consome um **único** valor, uma estrutura *posn*, e produz um único valor, um número.

Exemplos de uso de `distance-to-0`

```
(distance-to-0 (make-posn 0 5))  
= 5 and
```

```
(distance-to-0 (make-posn 7 0))  
= 7
```

A distância da origem de uma posição com coordenadas x e y é dada por $\sqrt{x^2 + y^2}$, logo

```
(distance-to-0 (make-posn 3 4))  
= 5
```

```
(distance-to-0 (make-posn 8 6))  
= 10
```

```
(distance-to-0 (make-posn 5 12))  
= 13
```

- Para acessar cada coordenada de um pixel, necessitamos de funções.
- Scheme fornece duas funções para acessar coordenadas de *posn*: `posn-x` e `posn-y`
- Exemplos que mostram relação dessas operações com `make-posn`

```
(posn-x (make-posn 7 0))  
= 7 and
```

```
(posn-y (make-posn 7 0))  
= 0
```

Podemos nos referir a uma estrutura *posn* por um nome:

```
(define point (make-posn 7 0))
```

E temos então:

```
(posn-x point)  
= 7
```

```
(posn-y point)  
= 0
```

```
(* (posn-x point) 7)  
= 49
```

```
(+ (posn-y point) 13) = 13
```

Podemos completar a definição de *distance-to-0*

```
(define (distance-to-0 a-posn)
  ... (posn-x a-posn) ...
  ... (posn-y a-posn) ...)
```

E finalmente chegamos a

```
(define (distance-to-0 a-posn)
  (sqrt
   (+ (sqr (posn-x a-posn))
      (sqr (posn-y a-posn)))))
```

Definição de Estrutura

- *usamos* estruturas *posn*
- *posn* combina 2 números e é útil para representar pixels
- Scheme permite programadores definir outras estruturas

- Definição da estrutura *posn*:

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

- após a avaliação dessa definição 3 operações são automaticamente criadas:
 - CONSTRUTOR `make-posn` que cria estruturas *posn*
 - SELETOR `posn-x` que seleciona coordenada *x*
 - SELETOR `posn-y` que seleciona coordenada *y*

- **Exemplo:** `(define-struct entry (name zip phone))`
- estrutura representa uma entrada no guia telefônico
- cada entrada combina os seguintes valores/campos:
 - nome,
 - cep, e
 - telefone

- 4 operações disponíveis:
 - `make-entry`,
 - `entry-name`,
 - `entry-zip`, **e**
 - `entry-phone`

Exemplo:

```
(make-entry 'PeterLee 15270 '606-7771)
```

cria uma estrutura *entry* com 'PeterLee no campo *name*, 15270 no campo *zip* e '606-7771 no campo *phone*

<i>name</i>	<i>zip</i>	<i>phone</i>
'PeterLee	15270	'606-7771

Exemplos:

```
(entry-name (make-entry 'PeterLee 15270 '606-7771))  
= 'PeterLee
```

Se dermos um nome para uma entrada:

```
(define phonebook (make-entry 'PeterLee 15270 '606-7771))
```

Podemos usar os seletores da seguinte forma:

```
(entry-name phonebook)  
= 'PeterLee
```

```
(entry-zip phonebook)  
= 15270
```

```
(define-struct star (last first instrument sales))
```

Após essa definição temos 5 operações:

- `make-star`,
- `star-last`,
- `star-first`,
- `star-instrument`, e
- `star-sales`

Criando estruturas `star`

```
(make-star 'Friedman 'Dan 'ukelele 19004)
```

```
(make-star 'Talcott 'Carolyn 'banjo 80000)
```

```
(make-star 'Harper 'Robert 'bagpipe 27860)
```

Exercício 6.3.1. Considere as seguintes definições de estruturas:

```
(define-struct movie (title producer))
```

```
(define-struct boyfriend (name hair eyes phone))
```

```
(define-struct cheerleader (name number))
```

```
(define-struct CD (artist title price))
```

```
(define-struct sweater (material size producer))
```

Quais são os nomes dos construtores e seletores criados?

Exercise 6.3.2. Considere a seguinte definição de estrutura:

```
(define-struct movie (title producer))
```

e avalie as seguintes expressões:

```
(movie-title (make-movie 'ThePhantomMenace 'Lucas))
```

```
(movie-producer (make-movie 'TheEmpireStrikesBack 'Lucas))
```

- funções podem consumir e produzir estruturas
- definir uma função que some 2000 as vendas de um `star`

```
;; increment-sales : star -> star  
;; dado um star produz o mesmo star  
;; com 2000 a mais em vendas
```

```
(define (increment-sales a-star) ...)
```

Exemplos de uso dessa função `increment-sales`

```
(increment-sales (make-star 'Abba 'John 'vocals 12200))
```

deve produzir

```
(make-star 'Abba 'John 'vocals 32200)
```

```
;; increment-sales : star -> star
;; dado um star produz o mesmo star
;; com 2000 a mais em vendas
```

```
(define (increment-sales a-star)
  (make-star (star-last a-star)
             (star-first a-star)
             (star-instrument a-star)
             (+ (star-sales a-star) 20000)))
```

Definição de Dados

Considere a seguinte expressão:

```
(make-posn 'Albert 'Meyer)
```

Ela gera uma estrutura *posn* a partir de 2 símbolos.

O que acontece se aplicarmos `distance-to-0` a essa estrutura?

```
(distance-to-0 (make-posn 'Albert 'Meyer))
= (sqrt
   (+ (sqr (posn-x (make-posn 'Albert 'Meyer)))
      (sqr (posn-y (make-posn 'Albert 'Meyer)))))
= (sqrt
   (+ (sqr 'Albert)
      (sqr (posn-y (make-posn 'Albert 'Meyer)))))
= (sqrt
   (+ (* 'Albert 'Albert)
      (sqr (posn-y (make-posn 'Albert 'Meyer)))))
ERRO NA EXECUÇÃO !!
```

Uma DEFINIÇÃO de DADOS estabelece, em uma mistura de Scheme e português

- como construir uma estrutura
- e como usá-la apropriadamente

Exemplo para *posn*

Um posn é uma estrutura

`(make-posn x y)`

onde x e y são números

Exemplo para estrutura *star*:

Um star é uma estrutura

`(make-star last first instrument sales)`

onde last, first, e instrument são símbolos e sales é um número.

- esperamos que programadores e usuários respeitem a definição de dados
- obviamente uma definição de dados não é uma garantia que a definição de dados será obedecida
- caso não seja seguida ocorrerá erro na execução

Exercício 6.4.1. Escreva definições de dados para as seguintes estruturas:

```
(define-struct movie (title producer))
```

```
(define-struct boyfriend (name hair eyes phone))
```

```
(define-struct cheerleader (name number))
```

```
(define-struct CD (artist title price))
```

```
(define-struct sweater (material size producer))
```

Exercício 6.4.2. Escreva a definição de uma estrutura para representar um ponto no tempo consistindo de de horas, minutos e segundos. Escreva também a definição de dados.

Exercício 6.4.3. Escreva a definição de estrutura e de dados para representar palavras de 3 letras. Uma palavra consiste de letras que são representadas com símbolos de ' a até ' z.

Projeto de Funções com Estruturas

Função que processa registro de estudantes. As informações importantes sobre estudantes são:

- primeiro nome
- último nome
- nome do professor titular

```
(define-struct student (last first teacher))
```

Definição de dados para *student*:

Um student é uma estrutura

```
(make-student l f t)
```

onde l, f, e t são símbolos.

Exemplos de estruturas *student*:

```
(make-student 'findler 'kathi 'matthias)
```

```
(make-student 'fisler 'sean 'matthias)
```

```
(make-student 'flatt 'shriram 'matthias)
```

O novo component no projeto é o *template* para funções que tem como entrada estruturas *student*

```
;; process-student : student ??? -> ???  
  
(define (process-student a-student ...)  
... (student-last a-student) ...  
... (student-first a-student) ...  
... (student-teacher a-student) ...)
```

Esse *template* pode ser usado para todas as funções que processam *student!*

Exercício 6.5.1. Desenvolva *templates* para funções que consomem estruturas abaixo:

```
(define-struct movie (title producer))
```

```
(define-struct boyfriend (name hair eyes phone))
```

```
(define-struct cheerleader (name number))
```

```
(define-struct CD (artist title price))
```

```
(define-struct sweater (material size producer))
```

Desenvolva uma função que, dados um *student* e o nome de um professor e produza o mesmo *student* trocando o nome do seu professor, caso esse nome seja 'Fritz' pelo nome do professor dado.

```
;; Definição e análise de dados:
```

```
(define-struct student (last first teacher))
```

```
;; Um student e uma estrutura: (make-student l f t)
```

```
;; onde f, l, e t são símbolos
```

```
;; Contrato: subst-teacher : student symbol -> student

;; Proposito: criar um student com novo
;; nome de professor, a-teacher, se o professor atual for 'Fritz

;; Exemplos:
;; (subst-teacher (make-student 'Find 'Matthew 'Fritz) 'Elise)
;; =
;; (make-student 'Find 'Matthew 'Elise)
;;
;; (subst-teacher (make-student 'Find 'Matthew 'Amanda) 'Elise)
;; =
;; (make-student 'Find 'Matthew 'Amanda)
```

```
;; Template:
;; (define (process-student a-student ???)
;; ... (student-last a-student) ...
;; ... (student-first a-student) ...
;; ... (student-teacher a-student) ...)

;; Definicao:
(define (subst-teacher a-student a-teacher)
  (cond
    [(symbol=? (student-teacher a-student) 'Fritz)
     (make-student (student-last a-student)
                    (student-first a-student)
                    a-teacher)]
    [else a-student]))
```

Exercício 6.5.2. Desenvolva a função `time->seconds`, que consome a estrutura para definição de tempo do exercício anterior e produz o número de segundos correspondentes desde a meia noite. Exemplo:

```
(time->seconds (make-time 12 30 2))  
= 45002
```