Definições Locais

Fundamentos de Algoritmos

INF05008

Escopo de Definições

- Linguagens de programação permitem que se definam nomes para:
  - Valores
  - Posições de memória
  - Funções e procedimentos
  - Objetos

- É importante saber onde os **nomes atribuídos são válidos** (**escopo**)

Definições

- Até agora, vimos algumas definições especiais de Scheme:
  - Funções: `(define (f a b) (+ a b))`
  - Valores: `(define x 42)`
  - Estruturas `(define-struct posn (x y))`
- Todas elas registram nomes que podem ser usados posteriormente:
  - `f, x, make-posn, posn-x, posn-y, posn?`

Ambiente de Nomes

- Um **ambiente de nomes** é uma tabela de definições que **associa cada nome a um objeto da linguagem**
  - `f` é a função “soma binária”.
  - `x` é o número 42.

- Inicialmente, o ambiente de nomes contém as definições **padrão** da linguagem.
  Ex: `+, -, empty, cons, pi,...`

- Cada comando de definição adiciona as definições do programador ao ambiente de nomes.
  Ex: `(define foo 77) => ...  + foo`

Escopo Global

- Após a definição, o nome pode ser **referenciado em várias partes do código**.

```
(define x 42)

(define (salário h) (* h 12))

(define (foo a)
  (+ a (salário x))
)
```

- `x`, `salário` e `foo` estão definidos **globalmente**

Escopo Global (cont.)

- Definições globais valem **até o final do programa**, **ou até serem redefinidas**.

```
(define lost 108)

(define (f x) (+ x lost))

(define lost 815)

(define (g x) (+ x lost))
```

- **A funções `f` e `g` *não* realizam a mesma computação!**

Escopo Local

- Considere as seguintes definições: (1)

```
(define x 42)

(define (salário h)
  (* h 12))

(define (foo a)
  (+ a (salário h)))
```

- O quê faz `foo`?

Escopo Local (cont.)

- Considere as seguintes definições: (2)

```
(define a 42)

(define (salário h)
  (* h 12))

(define (foo a)
  (+ a (salário a)))
```

- O quê faz `foo`?

Escopo Local (cont.)

- Na situação (1), a definição de `foo` não seria aceita pelo compilador: ele acusaria que não conhece `h`.

- Um **nome de parâmetro** só tem sentido **dentro do corpo da função correspondente** (**escopo local**). Como o nome `h` é um parâmetro de `salário`, ele não faz sentido dentro da definição de `foo`.

- Na situação (2), o comportamento de `foo` seria diferente do seu comportamento na situação (1). Em (2), `foo` utiliza somente o parâmetro recebido `a`, e não a definição global (que vale 42).

- **Quando um parâmetro possui o mesmo nome que uma definição global, ele "sobrescreve" esta definição.**

Definições Locais

- Há situações onde é interessante realizar definições dentro de um contexto específico:
  - Melhor organização de código
  - Evitar repetição de cálculos
- Scheme provê uma **sintaxe especial** para especificar **definições locais**

```
(local (def1
        def2
        ... )
 exp)
```

- Essa expressão resulta em `exp`, considerando `def1`, `def2`, ...

Definições Locais: Execução

- A forma

```
(local (def1 def2 ...)  exp)
```

executa da seguinte forma:

1. As definições são calculadas **na ordem**: `def1, def2 ...`
2. A expressão `exp` é calculada usando os **nomes locais**, e o resultado é **retornado**.
3. As definições locais calculadas são **descartadas**.

Definições Locais: Exemplo (1)

```
(define x 1)
(define y 2)
(define z 3)

(define u
     (local
       ( (define x 10)
         (define y 20) )
     (+ 4 x y z)))

(define v (+ x y))
```

- Qual é o valor de `u`? E o valor de `v`?

Definições Locais: Exemplo (2)

```
(define y 10)
(+ y
   (local ((define y 10)
   (define z (+ y y)))
     z))
```

- Qual é o valor final da função?

Definições Locais: Exemplo (3)

```
(define (soma x y)
  (local
       ( (define (soma-primeiro n) (+ x n)) )
       (soma-primeiro y)))
```

- Como é a execução de `soma`?

Definições locais: exemplo (4)

```
;; calcula-raízes: number number number -> lista-de-números
;; Cria uma lista com as raízes da equação quadrática
;; ax^2 + bx + c = 0
(define (calcula-raízes a b c)
  (local
       (
          (define delta (- (sqr b) (* 4 a c)))
          (define mb (- 0 b))
       )
     (cond
       [ (> delta 0) (list
                       (/ (+ mb (sqrt delta)) (* 2 a))
                       (/ (+ mb (- 0 (sqrt delta))) (* 2 a)))]
       [ (= delta 0) (list (/ mb (* 2 a)))]
       [ (< delta 0) empty])))
```

Uso de definições locais

- Quando uma função só existe para ser usada em um **contexto específico**, é interessante colocá-la em um **escopo local** (reforça a ideia de função auxiliar)
  - Ex: `insere` em `ordena`
- Quando alguma **subexpressão ocorre em mais de um ponto do corpo do programa**, pode ser interessante definir seu resultado como um **nome local** para não ter que repetir cálculos
  - Ex: `delta` em `calcula-raízes`

Exercícios

- Avalie as expressões abaixo:

```
(local ((define (x y) (* 3 y)))
 (* (x 2) 5))
```

```
(define (h n)
 (cond
 [(= n 0) empty]
 [else (local ((define r (* n n)))
                (cons r (h (- n 1))))]))
(h 5)
```

Exercícios

```
(define (f x z)
  (+
    (local (
      (define x 2)
      (define y 4))
        (local (
          (define x 10)
          (define z 3))
          (+ (* x y) z)))
    (* x z)))

(f 5 8)
```