Exercícios para

PROGRAMAÇÃO EM SCHEME:

Introdução à Programação Utilizando Múltiplos Paradigmas

Departamento de Engenharia Informática
Instituto Superior Técnico
Universidade Técnica de Lisboa

Conteúdo

1 Noções básicas **5**
- 1.1 Exercícios de revisão 5
- 1.2 Exercícios de programação 6

2 Procedimentos compostos **9**
- 2.1 Exercícios de revisão 9
- 2.2 Exercícios de programação 9
  - 2.2.1 Avaliação 9
  - 2.2.2 Procedimentos básicos 11
  - 2.2.3 Recursão 15
  - 2.2.4 Utilização de `let` 18
  - 2.2.5 Valores aproximados de funções 19
  - 2.2.6 Estrutura de blocos 20

3 Processos gerados por procedimentos **23**
- 3.1 Exercícios de revisão 23
- 3.2 Exercícios de programação 24

4 Procedimentos de ordem superior **29**
- 4.1 Exercícios de revisão 29
- 4.2 Exercícios de programação 29
  - 4.2.1 Procedimentos que recebem procedimentos 29
  - 4.2.2 Procedimentos que produzem procedimentos 31

5 Abstracção de dados **39**
- 5.1 Exercícios de revisão 39
- 5.2 Exercícios de programação 40
  - 5.2.1 Caixas e ponteiros 40
  - 5.2.2 Listas 40
  - 5.2.3 Árvores 48

5.2.4 Definição de novos tipos 49

6 O desenvolvimento de programas 59
6.1 Exercícios de revisão 59
6.2 Exercícios de programação 59

7 Programação imperativa 61
7.1 Exercícios de revisão 61
7.2 Exercícios de programação 61
7.2.1 Utilização da atribuição 61
7.2.2 Vectores 62
7.2.3 Procura e ordenação 65
7.2.4 Procedimentos com estado interno 66

8 Avaliação baseada em ambientes 73
8.1 Exercícios de revisão 73
8.2 Exercícios de programação 74

9 Estruturas mutáveis 79
9.1 Exercícios de revisão 79
9.2 Exercícios de programação 80
9.2.1 Definição de tipos mutáveis 87

10 Programação com objectos 89
10.1 Exercícios de revisão 89
10.2 Exercícios de programação 89

Capítulo 1

Noções básicas

1.1 Exercícios de revisão

1. O que é um processo computacional? Qual a relação entre um programa e um processo computacional?
2. O que é um algoritmo? Explique cada uma das suas propriedades: precisão, simplicidade e níveis de abstracção.
3. O que é um programa? Qual a sua relação com um algoritmo?
4. Dê um exemplo de cada um dos seguintes tipos de entidades existentes em Scheme
   (a) Constante
   (b) Procedimento primitivo
   (c) Combinação
   (d) Forma especial
5. Explique em que consiste o ciclo "lê-avalia-escreve".
6. O que é uma definição recursiva? Quais as partes que a compõem?
7. Em que consiste a abstracção? Qual a sua vantagem?
8. O que é a abordagem do topo para a base? Aplique-a à solução de um problema à sua escolha.
9. Diga o que é uma combinação, apresentando a sua sintaxe em notação BNF e explicando informalmente os símbolos não terminais que a compõem.

10. Quais os passos seguidos pelo Scheme na avaliação de uma combinação?

11. Considere a operação de nomeação em Scheme.

    (a) Qual a sintaxe e a semântica da operação de nomeação?

    (b) Explique a razão de esta operação corresponder a uma forma especial.

    (c) Qual a necessidade da definição desta operação?

12. O que é um predicado? Dê um exemplo.

13. O que é um ambiente?

1.2 Exercícios de programação

1. (a) Instale o Scheme no seu computador.

   (b) Na janela de interacção execute as seguintes acções:

   - Avalie a constante inteira correspondente ao seu número de aluno.
   - Avalie a cadeia de caracteres correspondente ao seu nome completo.
   - Calcule a diferença entre 2007 e o seu ano de nascimento.
   - Calcule a área de um círculo com diâmetro 10.
   - Calcule a razão entre a área de um quadrado com 10 de lado e a área de um círculo com diâmetro 10.

2. Traduza as seguintes expressões para notação prefixa e calcule o seu valor utilizando o interpretador do Scheme.

   (a) $(5 + 4 * 3)/(2 + 1)$

   (b) $(5 + 4) * 3/2 + 1$

   (c) $1 + 2 * 3/4 - 5 * 6/(7 + 8)$

3. Converta as seguintes expressões para a notação do Scheme:

   (a) $(1 + 5) * 6–12/3$

   (b) $2 + 3 * 4/5$

   (c) $(8 + 7 * 9/3 - 2) * (8/4 - 9 + 3)$

4. Avalie manualmente as seguintes formas. Indique todos os passos seguidos pelo avaliador do Scheme.

   (a) `(+ 2 5)`

   (b) `(* (- 5 3) 7)`

   (c) `(+ (* 2 9) (+ 2 3))`

   (d) `(* (/ 10 5) 7)`

5. Suponha que as seguintes expressões são avaliadas pelo Scheme. Na linha imediatamente a seguir a cada uma delas diga o que é escrito pelo interpretador do Scheme (se a avaliação de alguma das expresões der origem a um erro, explique a razão do erro).

   (a) `(+ 1 20)`

   (b) `(1 + 20)`

6. Suponha que as seguintes expressões são avaliadas pela ordem apresentada. Diga o que é escrito pelo interpretador do Scheme. Se a avaliação de alguma das expresões der origem a um erro, explique a razão do erro.

   (a) `(* 4 (+ 1 7.0))`

   (b) `(define a 5)`

   (c) `(+ a b)`

   (d) `(define b (* (+ 5 a) (+ 2 56)))`

   (e) `(+ a b)`

   (f) `a`

   (g) `(define a (+ a 2))`

   (h) `a`

   (i) `+`

   (j) 7.0

   (k) `(* 2 (/ 8 4))`

   (l) `(+ 3 #f)`

   (m) `(define a a)`

7. Diga qual o resultado de avaliar cada uma das seguintes formas. Considere que estas são fornecidas ao avaliador do Scheme pela ordem apresentada (e consequentemente cada forma é dependente das anteriores).

   Se a avaliação de uma forma não produzir resultados escreva `-nada-`. Se a avaliação de uma forma produzir um erro, explique a sua razão.

   (a) `(and (> 3 5) (/ 3 0))`

   (b) `(and (/ 3 0) (> 3 5))`

   (c) `(+ 8 (max 3 2))`

   (d) `(define a 5)`

Capítulo 2

Procedimentos compostos

2.1 Exercícios de revisão

1. Diga o que é abstracção procedimental e qual a sua vantagem.
2. Diga qual a diferença entre uma função matemática e um procedimento em Scheme para o cálculo dessa função.
3. Explique que são parâmetros formais e parâmetros concretos.
4. Utilizando a notação BNF, apresente a definição de uma expressão lambda. Explique cada um dos constituintes da sua definição.
5. O que é um procedimento anónimo? Dê um exemplo.
6. Escreva as regras seguidas pelo Scheme para a avaliação de uma expressão.
7. Diga qual é a regra de avaliação de uma expressão condicional.
8. Diga o que entende por linguagem estruturada em blocos. Descreva a regra associada a esta estrutura, e diga qual a sua importância.
9. O que entende pelo domínio de um nome?

2.2 Exercícios de programação

2.2.1 Avaliação

1. Avalie as seguintes formas. Se alguma das avaliações der origem a um erro, explique qual a razão do erro.

(a) `(3 1)`

(b) `((lambda (x) (+ 2 x)) 3)`

(c) `((lambda (x) ((lambda (y) (* y 2)) (+ x 3))) 4)`

2. Diga qual o resultado de avaliar cada uma das seguintes formas. Considere que estas são fornecidas ao avaliador do Scheme pela ordem apresentada (e consequentemente cada forma é dependente das anteriores).

   Se a avaliação de uma forma não produzir resultados escreva `-nada-`. Se a avaliação de uma forma produzir um erro, explique a sua razão.

   (a) `(define a 5)`

   (b) `(if (odd?  a) (+ a 2) (* a 3))`

   (c) `(lambda (x) (+ b x))`

   (d) `((lambda (x)(+ a x)) b)`

3. Considere o procedimento

```
(define (cont n)
   (if (< n 0)
       0
       (+ 2 (cont (- n 3)))))
```

   Com base nas regras de avaliação, desenhe a árvore que representa o funcionamento deste procedimento com a avaliação de `(cont 5)`.

4. Considere o procedimento

```
(define (proc n)
  (cond ((= n 0) 1)
        ((= n 1) 1)
        (else (+ 3 (proc (- n 1))))))
```

   Com base nas regras utilizadas pelo interpretador do Scheme para avaliar uma combinação, desenhe a árvore que representa o funcionamento deste procedimento com a avaliação de `(proc 3)`.

5. Repare que o nosso modelo de avaliação permite a existência de combinações cujos operadores são expressões compostas. Use esta observação para descrever o comportamento do seguinte procedimento:[1]

[1] Exercício 1.4 de Abelson H., Sussman G. J., e Sussman J., *Structure and Interpretation of Computer Programs*, 2^a Edição, p. 21, 1996.

```
(define (a-plus-abs-b a b)
  ((if (> b 0) + -) a b))
```

6. Considere a seguinte interacção em Scheme:

```
> (define (f1 x) (* x x))
> (f1 5)
25
> (define f2 f1)
```

   (a) Qual o valor de `(f2 5)`? Justifique a sua resposta.

   (b) Suponha que agora definia o procedimento `f2`, do seguinte modo:

   ```
   > (define (f2 x) (+ x 10))
   ```

   Quais os valores de `(f1 5)` e de `(f2 5)`? Justifique a sua resposta.

7. Considere a seguinte definição:

   `(define f ((lambda (x) (lambda (y) x)) 1))`.

   Diga quais os resultados da avaliação sequencial das seguintes expressões. Se alguma das expressões produzir um erro, explique a razão do erro.

   (a) `(f 2)`

   (b) `(define g (f 3))`

   (c) `(g 4)`

   (d) `(define h f)`

   (e) `(h 6)`

2.2.2 Procedimentos básicos

1. Escreva em Scheme um procedimento anónimo para representar a função de um argumento (x) cujo valor é calculado por $(x - 32) * 5/9$.

2. Escreva um procedimento anónimo para representar a função de dois argumentos (x e y) cujo valor é calculado por $(x + 3 * y) * (x–y)$.

3. Escreva um procedimento chamado `area-circulo` que calcula a área de um circulo cujo raio é r. Note-se que esta área é dada pela fórmula πr^2.

4. Utilizado o procedimento `area-circulo` do exercício anterior, escreva um procedimento que calcula a área de uma coroa circular de raio interior r_1 e raio exterior r_2.

5. Escreva em Scheme um procedimento anónimo, que recebe dois números e devolve o maior deles.

6. Escreva em Scheme um procedimento anónimo que soma dois ao seu argumento.

7. Escreva em Scheme o procedimento `dobro` que devolve o dobro do seu argumento.

8. Defina em Scheme o procedimento `valor-médio` que recebe dois números e devolve o número real que corresponde à média dos argumentos.

9. Escreva em Scheme o procedimento `metade` que recebe um número inteiro positivo e devolve um número racional que corresponde a metade do seu argumento.

10. Escreva um procedimento em Scheme chamado `minutos->horas` que recebe um inteiro correspondente a um certo número de minutos e que devolve um número real que traduz o número de horas correspondentes ao seu argumento.

    Por exemplo,

    ```
    > (minutos->horas 120)
    2.0
    > (minutos->horas 10)
    0.16666666666666666
    ```

11. Escreva um procedimento em Scheme chamado `cinco?` que tem o valor verdadeiro, apenas se o seu argumento for 5. Não utilize o `if`, nem o `cond`, nem `#t` nem `#f`.

12. Escreva um procedimento em Scheme `(entre? val x y)` que devolve `#t`, apenas no caso de `val` se encontrar entre `x` e `y` (`x` $<$ `val` $<$ `y`).

13. Escreva um procedimento em Scheme, chamado `sinal`, que receba um inteiro e devolva 1, -1 ou 0, caso o número seja, respectivamente, maior, menor ou igual a zero.

14. O procedimento primitivo `round`, quando a parte decimal do número a arredondar é 0.5, arredonda por excesso ou por defeito (em termos de valor absoluto), consoante a parte inteira seja ímpar ou par. Por exemplo,

```
>  (round 5.5)
6.0
>  (round 6.5)
6.0
```

Escreva um procedimento `arredonda` que arredonda sempre por excesso (em termos de valor absoluto), quando a parte decimal do número é 0.5. Por exemplo,

```
>  (arredonda 5.5)
6.0
>  (arredonda 6.5)
7.0
>  (arredonda -6.5)
-7.0
```

15. Escreva um procedimento em Scheme, chamado `quant->qual`, que recebe um real, representando uma nota quantitativa de um aluno (entre 0 e 20), e a transforma numa constante do tipo cadeia de caracteres que corresponde a uma nota qualitativa, de acordo com a tabela

Nota quantitativa	Nota qualitativa
18 a 20	Muito Bom
14 a 17	Bom
10 a 13	Suficiente
5 a 9	Mediocre
0 a 4	Mau

A nota recebida deverá ser convertida para um número inteiro, antes da conversão para a nota qualitativa.

16. Escreva um procedimento em Scheme que recebe as notas de um aluno de FP (nota do projecto, média dos trabalhos das práticas, nota do teste, e nota do exame final) e calcula a sua nota final de acordo com as seguintes regras:

    (a) A nota na cadeira será calculada por uma média ponderada da classificação obtida nas provas realizadas, com os seguintes pesos: Projecto, 0.25; Trabalhos de casa e aulas práticas, 0.10; Teste, 0.2; Exame final 0.45.

    (b) Para obter aprovação na cadeira, a nota do exame terá de ser superior a 7,5 valores, a nota do projecto terá de ser superior a 9,5 valores e a média ponderada da cadeira terá de ser superior a 9,5 valores.

    O seu procedimento deve devolver a nota final da cadeira (um inteiro) no caso de o aluno ter sido aprovado o zero no caso do aluno ter sido reprovado.

17. Utilizando os procedimentos desenvolvidos nos exercícios 15 e 16, escreva um procedimento em Scheme que recebe as notas de um aluno de FP (nota do projecto, média dos trabalhos das práticas, nota do teste, e nota do exame final) e calcula a sua nota qualitativa final.

18. Se a, b e c representam as dimensões dos lados de um triangulo, escreva um procedimento chamado `triangulo?` que devolve *verdadeiro* apenas se os valores de a, b e c poderem corresponder a lados de um triângulo: (a) nenhum dos valores a, b e c poderá ser negativo ou nulo; (b) a soma de quaisquer destes valores tem de ser maior do que o terceiro (num triângulo, qualquer lado é menor do que a soma dos outros dois).

19. Escreva um procedimento em Scheme que recebe três números correspondentes aos comprimentos dos lados de um triângulo e decide se o triângulo é equilátero, isósceles ou escaleno.

20. Se a, b e c representam as dimensões dos lados de um triangulo, a área do triângulo é dada por

$$\sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

    com

$$s = \frac{a+b+c}{2}$$

Escreva um procedimento em Scheme chamado `area-triangulo` que recebe como argumentos três inteiros representando as dimensões dos lados de um triângulo e que calcula a área do triângulo. Evite o cálculo repetido do valor de s. Verifique se as dimensões fornecidas correspondem a um triângulo.

2.2.3 Recursão

1. Defina um procedimento para somar os quadrados dos números naturais inferiores ou iguais ao seu argumento. Por exemplo, a avaliação de `(soma-quadrados 3)` origina $1^2 + 2^2 + 3^2$.

2. Escreva um procedimento em Scheme chamado `digitos->numero` que recebe como argumentos três dígitos e produz o número inteiro de três algarismos constituído pelos seus argumentos e pela ordem em que aparecem.

   Por exemplo,

   ```
   > (digitos->numero 3 2 8)
   328
   ```

3. Escreva um procedimento para calcular o número de dígitos de um número inteiro. Por exemplo,

   ```
   > (conta-digitos 76854)
   5
   ```

   Sugestão: utilize os procedimentos primitivos `quotient` e `remainder`, que calculam, respectivamente, a divisão inteira entre dois números e o resto da divisão inteira entre dois números.

4. Escreva um procedimento para calcular a soma dos dígitos de um número inteiro. Por exemplo,

   ```
   > (soma-digitos 12345)
   15
   ```

5. Escreva um procedimento `existe-digito?` que recebe um número inteiro e um dígito e tem o valor verdadeiro apenas se esse dígito existe no número. Por exemplo,

```
> (existe-digito 234 3)
#t
> (existe-digito 538 7)
#f
```

6. Escreva um procedimento que recebe um inteiro positivo e devolve um inteiro que é apenas constituído pelos dígitos impares do número original. Por exemplo,

   ```
   > (digitos-impares 1266458)
   15
   ```

   Assuma que o inteiro fornecido a este procedimento tem pelo menos um dígito impar.

7. O espelho de um número é o resultado de inverter a ordem de todos os seus algarismos. Por exemplo, o espelho de 391 é 139 e o espelho de 45679 é 97654. Escreva em Scheme um procedimento que receba um número inteiro e calcula o seu espelho. O procedimento não tem de verificar se o argumento recebido é um número inteiro.

8. Considere que a *ordem de um dígito* é a posição em que o dígito aparece num número. Considerando o número 5867, o dígito de ordem 1 é o 7 (o menos significativo), o dígito de ordem 2 é o 6, o de ordem 3 é o 8, o de ordem 4 é o 5 e os dígitos de ordem maior que 4 são 0.

   Defina o procedimento `digito-numero`, que recebe como argumentos a ordem de um dígito `ord` e um número inteiro não negativo `n` e devolve o dígito de ordem `ord` de `n`. Por exemplo,

   ```
   > (digito-numero 1 5867)
   7
   > (digito-numero 2 5867)
   6
   > (digito-numero 5 5867)
   0
   ```

9. Defina o procedimento `muda-digito-numero`, que recebe como argumentos a ordem de um dígito `ord`, um número inteiro não negativo `n` e um dígito `digito` e devolve um novo inteiro que tem os mesmos digitos nas mesmas ordens que `n` exceptuando o dígito de ordem `ord`, que passa a ser `digito`. Por exemplo,

```
> (muda-digito-numero 1 5867 2)
5862
> (muda-digito-numero 2 5867 2)
5827
> (muda-digito-numero 6 5867 2)
205867
```

10. Suponha que a operação `+` não existia em Scheme, mas que existiam os procedimentos `suc` e `ant` que recebiam como argumento um inteiro e devolviam, respectivamente, o seu sucessor e o seu antecessor. Escreva um procedimento chamado `mais` que recebe dois inteiros não negativos e devolve a sua soma. Use os procedimentos `suc` e `ant`. Para simplificação, pode admitir que ambos os inteiros a fornecer a este procedimento são positivos.

11. Considere que para além das condições estabelecidas no exercício , não existiam os predicados `>`, `<` nem `=`, existindo, no entanto o predicado `zero?` que tem o valor verdadeiro apenas se o seu argumento é zero.

    (a) Escreva um procedimento chamado `igual?` que recebe como argumentos dois números inteiros positivos e devolve *verdadeiro* apenas se estes forem iguais. Para simplificação, pode admitir que ambos os inteiros a fornecer a este procedimento são positivos.

    (b) Escreva um procedimento chamado `menor?` que recebe como argumentos dois números inteiros positivos e devolve *verdadeiro* apenas se o primeiro for menor que o segundo. Para simplificação, pode admitir que ambos os inteiros a fornecer a este procedimento são positivos.

12. Suponha que as operações `+`, `-`, `quotient`, `*` e `/` não existiam em Scheme, mas que existiam os procedimentos `suc` e `ant` que recebiam como argumento um inteiro e devolviam, respectivamente, o seu sucessor e o seu antecessor.

    Escreva um procedimento `div` para calcular a divisão inteira. A divisão inteira entre dois números inteiros a e b é o maior inteiro r tal que $r.b \leq a$. Por exemplo, `(div 5 2)` tem o valor 2, e `(div 9 3)` tem o valor 3.

    O seu procedimento apenas pode utilizar as operações `succ` e `ant`, podendo utilizar qualquer condição ou forma especial à sua escolha.

Para simplificação, pode admitir que ambos os inteiros a fornecer a este procedimento são positivos.

13. Escreva um procedimento em Scheme para verificar se um ano é bissexto. Um ano é bissexto se for divisível por 4 e não for divisível por 100, a não ser que seja também divisível por 400. Por exemplo, 1984 é bissexto, 1100 não é, e 2000 é bissexto.

14. Escreva um procedimento em Scheme para verificar se uma data é válida. O seu procedimento deve receber 3 inteiros, em que o primeiro representa o ano, o segundo, o mês, e o terceiro, o dia, e verificar se a data correspondente existe no calendário. Por exemplo, 1998 2 29 e 1945 4 31 não são datas válidas (porque 1998 não é bissexto e Abril só tem 30 dias, respectivamente) mas 2000 2 29 é uma data válida.

15. Um número inteiro é primo, se não for divísivel por nenhum inteiro menor que ele (excepto 1). Um modo, pouco eficiente, de determinar se um dado inteiro é primo corresponde a dividi-lo sucessivamente por todos os inteiros menores que ele. Escreva um procedimento `primo?` que recebe um número inteiro e determina se este é primo, utilizando o método descrito.

16. Suponha que existe definido o predicado `primo?` que, dado um número inteiro, indica se esse número é primo ou não. Defina o procedimento `nésimo-primo` que recebe um número inteiro `n` e devolve o enésimo número primo. Por exemplo,

```
>(nésimo-primo 1)
2
>(nésimo-primo 2)
3
>(nésimo-primo 7)
17
```

2.2.4 Utilização de `let`

1. (a) Qual o valor de

```
(let ((x 5))
  (let ((x 3)
        (y (* x 2)))
    (+ x y)))
```

(b) Traduza a expressão anterior para a utilização de uma expressão lambda.

2. Escreva a seguinte expressão, utilizando uma expressão `let`:

```
((lambda (x)
    ((lambda (y)
         (+ (* 2 y) x)) (+ x 3))) 5)
```

3. Suponha que `a` tem o valor 5. Diga qual o resultado de avaliar a expressão:

```
(let ((a 3)
      (b (* a 4)))
  (+ a b))
```

4. (a) Avalie a seguinte expressão

```
(let ((a 5)
      (b 12))
    (+ a (let ((a (+ a b)))
           (+ a 3))))
```

(b) Transforme a expressão anterior numa expressão lambda.

5. (a) Qual o resultado de avaliar a seguinte expressão?

```
((lambda (x y)
   (+ x y ((lambda (z)
             (* 2 z)) (+ y 1)))) 5 2)
```

(b) Escreva a expressão anterior usando o `let`.

2.2.5 Valores aproximados de funções

1. Suponha que o procedimento `sqrt` não estava definido em Scheme como um procedimento primitivo.

   A raiz quadrada de um número, x, pode ser calculada usando um método de aproximações sucessivas, chamado método de Newton, em que um valor aproximado para $\sqrt{x}$, y, é sucessivamente melhorado utilizando a fórmula $\frac{y+\frac{x}{y}}{2}$.

   Por exemplo, para calcular $\sqrt{3}$, poderíamos efectuar os seguintes cálculos:

Passo	Aproximação	Nova aproximação
1	3	2
2	2	1.75
3	1.75	1.732142857
4	1.732142857	1.73205081
5	1.73205081	1.732050808
6	1.732050808	1.732050808

Considerando que uma primeira aproximação grosseira para $\sqrt{x}$ pode ser x, escreva um procedimento em Scheme para calcular a raiz quadrada de um número.

2. Sabe-se que a seguinte fórmula permite calcular uma aproximação da função *seno* até um determinado número de termos:

$$seno(x) = x(1 - \frac{x^2}{2 \cdot 3}(1 - \frac{x^2}{4 \cdot 5}(1 - \frac{x^2}{6 \cdot 7}(\cdots))))$$

Defina em Scheme um procedimento `sin` que corresponde ao método de cálculo anterior. O seu procedimento deverá receber o número para o qual se pretende calcular o seno e o número de factores a considerar.

2.2.6 Estrutura de blocos

1. Considere as seguintes alternativas para definir um procedimento que calcula a função factorial.

Alternativa 1:

```
(define (factorial n)
  (if (> n 0)
      (fact-aux n)
      "erro argumento negativo ou nulo"))

(define (fact-aux n)
  (if (= n 1)
      1
      (* n (fact-aux (- n 1)))))
```

Alternativa 2:

```
(define (factorial n)
  (define (fact-aux n)
    (if (= n 1)
        1
        (* n (fact-aux (- n 1)))))
  (if (> n 0)
      (fact-aux n)
      "erro argumento negativo ou nulo"))
```

(a) Compare estas duas alternativas quanto à garantia de que o factorial não é calculado com valores indevidos. SUGESTÃO: será que se pode avaliar directamente `(fact-aux -1)`?

(b) Como se chama a estrutura definida pela alternativa 2? Qual a sua vantagem?

2. A estrutura de blocos leva à existência de novos ambientes. Com base no procedimento que se segue, que calcula $m! - n!$, justifique a afirmação. Sugestão: identifique as variáveis locais e não locais para cada um dos procedimentos.

```
(define (diferenca-de-factoriais m n)
  (define (iter m res)
    (if (= m n)
        res
        (iter (- m 1)(* m res))))
  (iter m 1))
```

Capítulo 3

Processos gerados por procedimentos

3.1 Exercícios de revisão

1. Explique os seguintes conceitos: procedimento e processo gerado por um procedimento.

2. Quais são as características de um processo recursivo?

3. Quais são as características de um processo iterativo linear?

4. Será que um procedimento recursivo pode gerar um processo iterativo? Justifique a sua resposta e dê um exemplo.

5. Considere a afirmação "os procedimentos que geram processos iterativos são sempre melhores que os procedimentos que geram processos recursivos". Comente-a relativamente aos seguintes aspectos: (a) memória ocupada; (b) tempo gasto; (c) facilidade de escrita e leitura do programa.

6. Qual o interesse de estudar a ordem de crescimento de um processo? Compare as ordens de crescimento (temporal e espacial) entre processos recursivos lineares e iterativos lineares.

7. Diga o que significa a frase "A ordem de crescimento do recurso $R(n)$ é $\Theta(n)$".

8. Usando a definição da função Θ que define a ordem de crescimento de um processo, explique o que significa um processo ter ordem de crescimento polinomial.

3.2 Exercícios de programação

1. Defina um procedimento recursivo que recebe três argumentos (sejam `a`, `b` e `n`, por exemplo) e que devolve o valor de somar `a` a `b` `n` vezes.

   (a) Gerando um processo recursivo.

   (b) Gerando um processo iterativo

2. Considere a definição do seguinte procedimento em Scheme:

```
(define (misterio x y)
  (define (misterio-aux m1 m2 r)
    (if (= m2 0)
        r
        (misterio-aux m1 (- m2 1) (+ m1 r))))
  (misterio-aux x y 0))
```

   (a) Mostre a evolução do processo originado pela execução de `(misterio 7 3)`.

   (b) O processo gerado é recursivo ou iterativo? Explique.

   (c) O procedimento `misterio-aux` é recursivo? Justifique a sua resposta.

3. Considere a definição do seguinte procedimento em Scheme que recebe inteiros superiores ou iguais a zero:

```
(define (xpto? a)
  (define (aux b c)
    (cond ((zero? b) #t)
          ((zero? c) #f)
          (else (aux (sub1 (sub1 b))
                     (sub1 (sub1 c))))))
  (aux a (sub1 a)))
```

   (a) Explique que função calcula este procedimento.

(b) O processo gerado pelo procedimento `aux` é iterativo ou recursivo? Justifique a sua resposta.

4. Considere o seguinte procedimento:

```
(define (misterio n)
  (define (mist-aux n ac)
    (if (< n 10)
        (+ n (* ac 10))
        (mist-aux (quotient n 10)
                  (+ (remainder n 10) (* ac 10)))))
  (mist-aux n 0))
```

(a) Entre os procedimentos apresentados, `misterio` e `mist-aux`, existe algum que seja um procedimento recursivo? Justifique a sua resposta.

(b) Mostre a evolução do processo gerado pela avaliação `(misterio 149)`.

(c) O procedimento `misterio` gera um processo recursivo ou iterativo? Justifique a sua resposta.

5. Considere o seguinte procedimento

```
(define (misterio x n)
  (if (= n 0)
      0
      (+ (* x n) (misterio x (- n 1)))))
```

(a) Mostre a evolução do processo gerado pela avaliação de `(misterio 2 3)`.

(b) O procedimento apresentado é um procedimento recursivo? Justifique.

(c) O procedimento apresentado gera um processo recursivo ou iterativo? Justifique.

(d) Se o processo gerado era iterativo, transforme o procedimento de forma que gere um processo recursivo. Se o processo gerado era recursivo, transforme o procedimento, de forma a que gere um processo iterativo.

6. O procedimento

```
(define (factorial n)
    (if (= n 0)
        1
        (* n (factorial (- n 1)))))
```

calcula o factorial através de multiplicações da esquerda para a direita ($5! = 5 \times 4 \times 3 \times 2 \times 1 = 20 \times 3 \times 2 \times 1 = 60 \times 2 \times 1 = 120 \times 1 = 120$).

Em alternativa, poderíamos ter efectuado as multiplicações da direita para a esquerda ($5! = 5 \times 4 \times 3 \times 2 \times 1 = 5 \times 4 \times 3 \times 2 = 5 \times 4 \times 6 = 5 \times 24 = 120$).

Escreva um procedimento que gera um processo iterativo e que calcula o valor de factorial multiplicando da direita para a esquerda. Note que vai ter de usar um parâmetro adicional para o cálculo de factorial auxiliar.

7. (a) Escreva um procedimento em Scheme para calcular o valor da função de Ackermann:[1]

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(m-1,1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1, A(m,n-1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

   (b) Siga o processo gerado pelo cálculo de $A(2,2)$.

8. Escreva um procedimento que gera um processo iterativo e que corresponde à realização da seguinte função:

$$x^n = \begin{cases} x & \text{se } n = 1 \\ x.(x^{n-1}) & \text{se } n \text{ for ímpar} \\ (x^{n/2})^2 & \text{se } n \text{ for par} \end{cases}$$

9. Suponha que não existe o operador `*` em Scheme (mas que existem os procedimentos adição `+` e divisão `/` e um predicado que identifica números pares `even?`). Um modo de definir o produto de dois números inteiros positivos é através da seguinte fórmula:

$$x.y = \begin{cases} 0 & \text{se } x = 0 \\ \frac{x}{2}.(y+y) & \text{se } x \text{ é par} \\ y + (x-1).y & \text{em caso contrário} \end{cases}$$

[1] Tal como apresentada em [Machtey e Young 78], pág. 24.

(a) Defina o procedimento `prod` que efectua o produto de dois números inteiros positivos, de acordo com esta definição.

(b) O seu procedimento é um procedimento recursivo? Justifique.

(c) O seu procedimento gera um processo recursivo ou iterativo? Justifique a sua resposta.

(d) Se o processo gerado era iterativo, transforme o procedimento, de forma a que gere um processo recursivo. Se o processo gerado era recursivo, transforme o procedimento, de forma a que gere um processo iterativo.

10. O número de combinações de m objectos n a n pode ser dado pela seguinte fórmula:

$$C(m,n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } m = n \\ C(m-1,n) + C(m-1,n-1) & \text{se } m > n,\ m > 0 \text{ e } n > 0 \end{cases}$$

(a) Escreva um procedimento em Scheme para calcular o número de combinações de m objectos n a n.

(b) Que tipo de processo é gerado pelo seu procedimento?

11. O que faz o seguinte procedimento em Scheme?

```
(define (misterio x)
  (cond ((= x 0) (newline))
        (else (display (remainder x 10))
              (misterio (quotient x 10)))))
```

Sugestão: siga o processo gerado por este procedimento para alguns inteiros. Depois admire o poder dos procedimentos recursivos.

Capítulo 4

Procedimentos de ordem superior

4.1 Exercícios de revisão

1. Diga quando é que em programação se diz que um objecto computacional é um cidadão de primeira classe.

2. Diga o que é um procedimento de ordem superior e quais são as suas vantagens.

4.2 Exercícios de programação

4.2.1 Procedimentos que recebem procedimentos

1. O procedimento `somatorio`

```
(define (somatorio calc-termo lim-inf proximo lim-sup)
  (if (> lim-inf lim-sup)
      0
      (+ (calc-termo lim-inf)
         (somatorio calc-termo
                    (proximo lim-inf)
                    proximo
                    lim-sup))))
```

   é apenas o mais simples de um vasto número de abstracções semelhantes que podem ser capturadas por procedimentos de ordem superior.

(a) Defina um procedimento chamado `piatorio` que calcula o produto dos termos de uma função entre dois limites especificados.

(b) Mostre como definir o factorial em termos da utilização do procedimento `piatorio`.

2. O procedimento `produto` devolve o produto dos valores de uma função, `fn`, para pontos pertencentes a um intervalo.

```
(define (produto fn a prox b)
   (if (> a b)
       1
       (* (fn a) (produto fn (prox a) prox b))))
```

Use o procedimento `produto` para definir o procedimento *sen* que calcula o seno de um número, usando a seguinte aproximação:

$$sen(x) = x(1 - (\frac{x}{\pi})^2)(1 - (\frac{x}{2\pi})^2)(1 - (\frac{x}{3\pi})^2)\cdots$$

O seu procedimento deverá receber o número para o qual se pretende calcular o seno e o número de factores a considerar.

3. Dada uma função f e dois pontos a e b do seu domínio, um dos métodos para calcular a área entre o gráfico da função e o eixo dos xx no intervalo $[a,\ b]$ consiste em dividir o intervalo $[a,\ b]$ em n intervalos de igual tamanho, $[x_0,\ x_1]$, $[x_1,\ x_2]$, ..., $[x_{n-2},\ x_{n-1}]$, $[x_{n-1},\ x_n]$, com $x_0 = a$, $x_n = b$, e $\forall i, j,\ x_i - x_{i-1} = x_j - x_{j-1}$. A área sob o gráfico da função será dada por (Figura 4.1):

$$\sum_{i=1}^{n} f\left(\frac{x_i + x_{i-1}}{2}\right)\ (x_i - x_{i-1})$$

Escreva em Scheme um procedimento de ordem superior que recebe como argumentos um procedimento (correspondente à função f) e os valores de a e b e que calcula o valor da área entre o gráfico da função e o eixo dos xx no intervalo $[a,\ b]$.

O seu procedimento poderá começar a calcular a área para o intervalo $[x_0,\ x_1] = [a,\ b]$ e ir dividindo sucessivamente os intervalos $[x_{i-1},\ x_i]$ ao meio, até que o valor da área seja suficientemente bom.

4. Suponha que `f` é um procedimento em Scheme que corresponde à realização de uma função estritamente crescente de números naturais

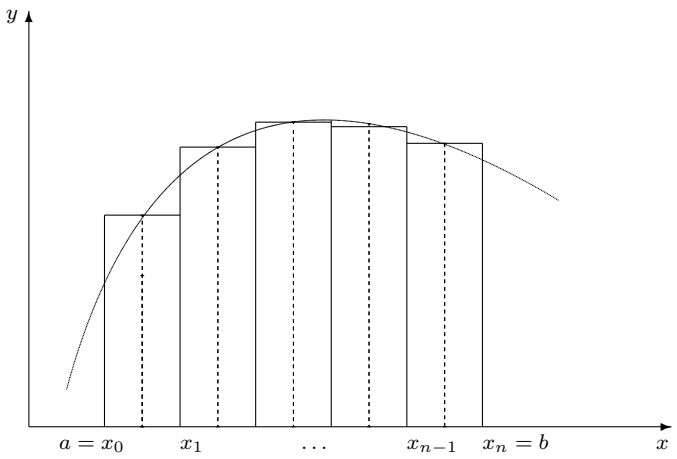


Figura 4.1: Área aproximada sob a curva.

para números naturais. Escreva um procedimento em Scheme chamado `procura` que recebe como argumentos o procedimento `f`, um valor `objectivo` e dois valores do domínio de `f`, `inicial` e `final` e determina se o valor `objectivo` se encontra entre a sequência de valores `(f inicial)` ... `(f final)`. No caso de este se encontrar, o valor do procedimento procura é o `x` tal que `(f x)` = `objectivo`, em caso contrário o valor do procedimento procura é `0`.

Por exemplo, `(procura (lambda(x) (* x x)) 25 1 10)` tem o valor 5 e `(procura (lambda(x) (* x x)) 12 1 10)` tem o valor 0. Sugestão: utilize uma técnica semelhante à utilizada no cálculo de raízes pelo método do intervalo.

4.2.2 Procedimentos que produzem procedimentos

1. A composição de duas funções de um argumento, $f(x)$ e $g(x)$, é a função $f(g(x))$.

   Escreva um procedimento em Scheme, chamado `fn-composta`, que recebe como argumentos dois procedimentos correspondentes a funções de um argumento e que devolve o procedimento correspondente à composição dos seus argumentos.

   Por exemplo, com o seu procedimento, seria gerada a seguinte inte-

racção:

```
> (define (quadrado x) (* x x))
> (define (soma-6 y)(+ 6 y))
> ((fn-composta quadrado soma-6) 3)
81
```

2. (a) Escreva um procedimento em Scheme chamado `fn-soma` que receba como argumentos dois procedimentos correspondentes a funções reais de variável real, f e g, e devolva o procedimento correspondente à *função* correspondente à soma de f com g.

   Por exemplo, com este procedimento podemos gerar a interacção:

   ```
   > (define (f1 x) (+ x 3))
   > (define (f2 y) (* y 10))
   > ((fn-soma f1 f2) 12)
   135
   ```

   (b) Podemos pensar numa generalização do procedimento da alínea anterior, fornecendo a operação a realizar entre as funções f e g. Por exemplo, se a operação for a adição o procedimento corresponde ao anterior devolvendo $f(x) + g(x)$; no entanto, se a operação for a potência o procedimento devolve $f(x)^{g(x)}$.

   Escreva um procedimento em Scheme, chamado `aplica-op` que recebe como argumentos uma operação (aplicável a dois argumentos) e duas funções reais de variável real e que devolve o procedimento que corresponde a aplicar a operação fornecida às duas funções.

   Por exemplo,

   ```
   > (define (f1 x) (+ x 3))
   > (define (f2 y) (* y 10))
   > ((aplica-op + f1 f2) 12)
   135
   > ((aplica-op * f1 f2) 12)
   1800
   ```

3. Defina um procedimento de ordem superior que recebe procedimentos para calcular as funções reais de variável real f e g e que se comporta como a seguinte função matemática:

$$h(x) = f(x)^2 + 4g(x)^3$$

4. Escreva um procedimento, chamado `nega`, que recebe como argumento um predicado de dois argumentos e que devolve o procedimento correspondente à negação do seu argumento. Por exemplo,

```
> ((nega >) 2 3)
#t
> ((nega >) 2 2)
#t
> ((nega <=) 4 5)
#f
```

5. Escreva o procedimento chamado `conjunção`, que recebe dois predicados de um argumento cada, e devolve um novo predicado de um argumento que verifica se esse argumento satisfaz ambos os predicados recebidos originalmente. Por exemplo,

```
> (define par>5? (conjunção even? (lambda (x) (> x 5))))
> (par>5? 4)
#f
> (par>5? 8)
#t
> (par>5? 9)
#f
```

6. (a) Escreva um procedimento chamado `arredonda`, que recebe um inteiro n, e devolve um procedimento que recebe um argumento real e o arredonda para n casas decimais.

   (b) Utilize o procedimento `arredonda` para definir o procedimento `arredonda-3`, que recebe um real e o arredonda para 3 casas decimais. Por exemplo,

```
> (arredonda-3 1.23456)
   1.235
```

7. Escreva um procedimento que recebe um número e devolve um procedimento que recebe outro número e o multiplica pelo primeiro. Por exemplo,

```
> (define mult5 (cria-multiplicador 5))
> (mult5 2)
10
```

Este exercício é um exemplo da definição de um procedimento de dois argumentos como um procedimento de um argumento, cujo valor é um procedimento de um argumento. Esta técnica, que é útil quando desejamos manter um dos argumentos fixo, enquanto o outro pode variar, é conhecida como método de "*Curry*", e foi concebida por Moses Schönfinkel[1] e baptizada em honra do matemático Haskell B. Curry.

8. Escreva o procedimento `faz-potencia` que recebe, como argumento, um inteiro não negativo e devolve um procedimento de um argumento, `n`, que calcula a potência `n` desse inteiro.

   Por exemplo, `(faz-potencia 3)` devolve o procedimento que calcula o cubo do seu argumento.

9. Considere as seguintes definições para um procedimento que devolve a primeira derivada de uma função:

```
(define (derivada f)
  (lambda (x)
    (/ (- (f (+ x dx))(f x))
       dx)))

(define dx 0.00001)
```

   Com base na definição anterior escreva um procedimento que recebe um procedimento correspondente a uma função e um inteiro n ($n \geq 1$) e devolve a derivada de ordem n da função. A derivada de ordem n de uma função é a derivada da derivada de ordem $n-1$. Utilize o procedimento `derivada`.

10. Tendo em atenção que $\sqrt{x}$ é o número y tal que $y^2 = x$, ou seja, para um dado x, o valor de $\sqrt{x}$ corresponde ao zero da equação $y^2 - x = 0$, utilize o método de Newton para escrever um procedimento que calcula a raiz quadrada de um número.

11. Escreva um procedimento chamado `rasto`, que recebe uma cadeia de caracteres correspondendo ao nome de um procedimento, e um procedimento de um argumento. O procedimento rasto retorna um procedimento de um argumento que escreve no écran (usando o procedimento primitivo `display`) a indicação de que o procedimento foi

[1] Schönfinkel M., "On the building blocks of Mathematical Logic", em *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931*, Jean van Heijenoort (ed.), Cambridge, MA: Harvard University Press, 1977.

avaliado e o valor do seu argumento, escreve também o resultado do procedimento, e devolve o valor de aplicar o procedimento original ao argumento recebido. Por exemplo, partindo do princípio que o procedimento `quadrado` foi definido, podemos gerar a seguinte interacção:

```
> (define rasto-quadrado (rasto "quadrado" quadrado))
> (rasto-quadrado 4)
Avaliação de quadrado, com o argumento 4
Resultado 16
16
>
```

12. A sequência de Fibonacci (ou números de Fibonacci) é definida recursivamente do seguinte modo:

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{se } n > 1 \end{cases}$$

ou seja, depois dos dois termos iniciais, cada termo da sequência corresponde à soma dos dois termos anteriores.

Sequências semelhantes à de Fibonacci, mas com termos iniciais diferentes, têm sido descobertas como transcrevendo vários aspectos da natureza. Um dos exemplos, a sequência de Lucas, em honra ao Matemático francês François Lucas (1842–1891), corresponde à sequência 3, 1, 4, 5, 9, ...

A sequência de Fibonacci pode ser generalizada a uma colecção de sequências, parametrizadas com base nos dois primeiros termos, a e b e definidas através da seguinte expressão:

$$F_{a,b}(n) = \begin{cases} a & \text{se } n = 0 \\ b & \text{se } n = 1 \\ F_{a,b}(n-1) + F_{a,b}(n-2) & \text{se } n > 1 \end{cases}$$

Assim, $F_{0,1}(n)$ é uma expressão designatória correspondente ao n-ésimo termo da sequência de Fibonacci, $F_{3,1}(n)$, é uma expressão designatória correspondente ao n-ésimo termo da sequência de Lucas.

(a) Defina um procedimento em Scheme que dê origem a um procedimento que calcula o n-ésimo termo da sequência generalizada de

Fibonacci. O seu procedimento deve receber como argumentos os dois primeiros termos da sequência, produzindo um procedimento de um argumento que calcula o n-ésimo termo de uma sequência particular. O procedimento deve gerar um processo iterativo.

(b) Use o procedimento da alínea anterior para gerar outro procedimento que calcula a sequência generalizada de Fibonacci cujos dois primeiros termos são o ano do seu nascimento e o seu número de aluno. Usando este procedimento, calcule o termo cuja posição corresponde a 10 somado ao mês do seu nascimento. Por exemplo, um aluno nascido em Março de 1988 e cujo número é 12345 deverá calcular o valor de $F_{1988,12345}(13)$.

13. Um dos aspectos interessantes associados às sequências de Fibonacci generalizadas corresponde ao facto de, independentemente dos valores de a e b, o limite da razão entre um termo e o anterior quando n tende para infinito é sempre o mesmo:

$$\lim_{n\to\infty} \frac{F_{a,b}(n+1)}{F_{a,b}(n)} = \varphi$$

em que φ (leia-se PHI, o qual não deve ser confundido com o número π) é dado por

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180$$

Phi é um número transcendente conhecido por *proporção divina*, *proporção áurea* ou *número de ouro*. A proporção divina foi recentemente glorificada no livro de Dan Brown, *O Código da Vinci*. Este número, frequentemente usado nas proporções das pinturas renascentistas, está envolvido com a natureza do crescimento. Phi pode ser encontrado na proporção em conchas, seres humanos, e até na relação entre os machos e fêmeas de qualquer colmeia do mundo. Por esta razão, os antigos consideravam que este número tinha sido utilizado pelo Criador do universo e chamavam-lhe a *proporção divina*. Esta proporção corresponde à relação que resulta quando uma linha é dividida de tal modo que o comprimento total da linha tem a mesma relação com o comprimento do segmento maior que o comprimento do segmento maior tem com o comprimento do segmento menor.

Usando um método de aproximações sucessivas semelhante ao apresentado na Secção 2.4.3 do livro, escreva em Scheme um procedimento de ordem superior que calcule o valor da proporção divina com uma precisão de 10 casas decimais. O argumento do seu procedimento deverá ser um procedimento para o cálculo da sequência de Fibonacci. O seu procedimento deverá utilizar expressões "`let`"para evitar cálculos repetidos.

Capítulo 5

Abstracção de dados

5.1 Exercícios de revisão

1. Diga o que é um tipo de informação.
2. Qual a diferença entre tipos de informação elementares e tipos de informação estruturados.
3. Explique em que consiste a abstracção de dados, usando os termos barreiras de abstracção, encapsulação da informação e anonimato da representação.
4. Quais as vantagens da abstracção de dados.
5. Compare a abstracção de dados com a abstracção procedimental.
6. Justifique a seguinte afirmação "a abstracção de dados aumenta o poder expressivo da nossa linguagem de programação".
7. Quais os quatro passos a seguir na definição de um tipo abstracto de informação? Explique em que consiste cada um deles.
8. Diga o que são operações básicas de um tipo abstracto de informação e quais os grupos em que estas são divididas.
9. Explique o que são barreiras de abstracção criadas por um tipo de informação e quais os inconvenientes de não as respeitar.
10. Diga o que é o anonimato da representação e qual a sua importância.
11. Na criação de um tipo abstracto de informação, há dois tipos de representação a considerar. Diga quais são e descreva cada uma deles.

12. Diga o que é uma árvore binária e quais são as suas operações básicas, classificando-as de acordo com os vários grupos de operações.

5.2 Exercícios de programação

5.2.1 Caixas e ponteiros

1. Represente as seguintes estruturas, usando a notação de caixas e ponteiros:

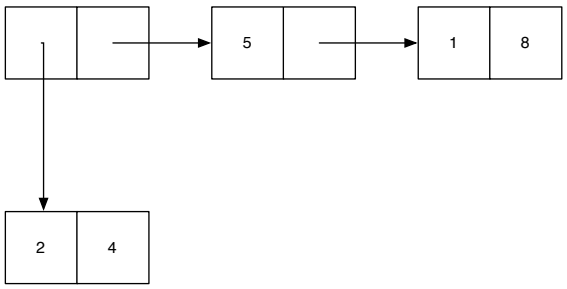
   (a) `(1 . 2)`

   (b) `(1 ("bom dia". 3))`

   (c) `(1 (2 . (3 . 4)) 5)`

   (d) `((1 . 2) ((((3 (4 . 5) 6) (7 . 8)))))`

2. Considere a seguinte estrutura de pares (correspondente ao nome `estrutura`):



   Para cada uma das seguintes expressões escreva o seu valor (o qual pode ser um número inteiro, uma estrutura de pares ou pode originar um erro). No caso de a expressão originar um erro explique a razão do erro.

   (a) `(car estrutura)`

   (b) `(car (car (cdr estrutura)))`

   (c) `(cdr (cdr (cdr estrutura)))`

5.2.2 Listas

1. Usando a representação de listas simplificadas baseadas em pares, mostre as estruturas de pares (usando a representação gráfica de caixas e ponteiros) correspondentes às seguintes listas:

(a) `(1 ((2 3) 4) 5)`

(b) `(1 2 3 ((4 (5))))`

(c) `(1 (2 (3)) ((4)) 5)`

2. (a) Escreva um procedimento que receba como argumento um inteiro positivo e devolva uma lista com todos os inteiros entre 0 e esse inteiro (a ordem pela qual os números aparecem na lista não é relevante).

   (b) O seu procedimento é um procedimento recursivo? Porquê?

   (c) O seu procedimento gera um processo iterativo ou recursivo? Porquê?

   (d) Se o seu procedimento gera um processo iterativo, escreva um que gere um processo recursivo. Se gera um processo recursivo, escreva um que gere um processo iterativo.

3. Escreva um procedimento que receba uma lista não vazia e devolva o seu último elemento.

4. Defina um procedimento chamado `otser` (resto ao contrário) que recebe como argumento uma lista e devolve a lista com todos os elementos da lista original excepto o último. O seu procedimento deve gerar uma mensagem de erro no caso da lista ser vazia. O seu procedimento deve respeitar as barreiras de abstracção. Por exemplo,

```
> (otser ‘(3 1 7 5))
(3 1 7)
> (otser ‘(6))
()
> (otser ‘())
otser: a lista não tem elementos
```

5. Defina um procedimento chamado `eresni` (insere ao contrário) que recebe como argumentos um elemento e uma lista e insere o elemento no fim da lista. O seu procedimento deve respeitar as barreiras de abstracção. Por exemplo,

```
> (eresni 2 ‘(3 1 7 5))
(3 1 7 5 2)
> (eresni 6 ‘())
(6)
```

6. Defina o procedimento `nem-todos?` que recebe um predicado e uma lista e tem como valor *verdadeiro*, se existir pelo menos um elemento na lista que não satisfaz o predicado, e $falso$, caso contrário. Por exemplo,

```
>(nem-todos? even? '(2 4 5 6))
#t
>(nem-todos? even? '(2 4 6))
#f
```

7. Defina o procedimento `ordenada?` que recebe um predicado de dois argumentos e uma lista e tem como valor *verdadeiro*, se a lista se encontra ordenada de acordo com o predicado recebido, e $falso$ caso contrário. Por exemplo,

```
>(ordenada? < '(1 2 3))
#t
>(ordenada? > '(1 2 3))
#f
>(ordenada? >= '(3 3 2 2))
#t
```

8. Defina o procedimento `duplica` que recebe uma lista e que tem como valor uma lista idêntica à original, mas em que cada elemento está repetido. Por exemplo,

```
>(duplica '(1 2 3))
(1 1 2 2 3 3)
```

   O seu procedimento que escreveu gera um processo recursivo ou iterativo? Justifique a sua resposta.

9. Defina um procedimento `juntos` que recebe uma lista de inteiros e tem como valor o número de elementos iguais adjacentes. Por exemplo,

```
>(juntos '(1 2 2 3 4 4))
2
>(juntos '(1 2 3 4))
0
```

10. Defina o procedimento `conta-menores` que recebe um inteiro e uma lista de inteiros e devolve o número de elementos dessa lista que são menores que o inteiro recebido. Por exemplo,

```
>(conta-menores 5 '(2 7 4 8 5))
2
```

    (a) Usando um processo recursivo.

    (b) Usando um processo iterativo.

11. (a) Escreva um procedimento `insere-por-ordem` que recebe um inteiro e uma lista de inteiros, ordenada por ordem crescente, e devolve a lista resultante de inserir o elemento na lista recebida, de forma a continuar a ter uma lista ordenada. Por exemplo,

```
> (insere-por-ordem  2 '(1 3 5))
(1 2 3 5)
> (insere-por-ordem  3 '(1 3 5))
(1 3 3 5)
```

    (b) O procedimento que escreveu é recursivo? E o processo gerado? Justifique as suas respostas.

    (c) Se o seu procedimento gera um processo recursivo, escreva um procedimento equivalente que gera um processo iterativo, e vice-versa.

12. O João definiu o seguinte procedimento:

```
(define (conta-menores n l)
  (if (< (primeiro l) n)
      (+ 1 (conta-menores n (resto l)))
      (conta-menores n (resto l))))
```

    (a) O João está confiante de que obterá o seguinte resultado:

```
> (conta-menores 5 '(1 3 5 7))
2
```

    Na realidade, o que vai acontecer? Porquê?

    (b) Volte a definir o procedimento de modo a que funcione correctamente.

13. Escreva um procedimento chamado `filtre`, que recebe como argumentos um predicado e dois números naturais `m` e `n` ($\texttt{m} \leq \texttt{n}$) e que devolve a lista de todos os números naturais entre `m` e `n` que satisfazem o predicado. Por exemplo, partindo do pressuposto da existência dos predicados `par?` e `primo?` que testam, respectivamente, se um número é par ou é primo, obtemos o seguinte comportamento

```
> (filtre par? 1 10)
(2 4 6 8 10)
> (filtre primo? 10 20)
(11 13 17 19)
```

14. Escreva um procedimento que recebe, no mínimo, um argumento inteiro e que devolve a lista que se obtém somando o primeiro argumento a cada um dos restantes elementos. Por exemplo,

```
> (aumenta 10 2 5 8)
(12 15 18)
> (aumenta 8 20)
(28)
> (aumenta 3)
()
> (aumenta)
procedure aumenta: expects at least 1 argument, given 0
```

15. Escreva um procedimento chamado `capicua` que recebe uma lista e tem como valor uma lista que se obtém construindo uma capicua com a lista inicial. Por exemplo:

```
>(capicua ‘(3 4 7 1))
(3 4 7 1 1 7 4 3)
>(capicua ‘(1 2))
(1 2 2 1)
```

16. Escreva um procedimento `conta` que recebe um elemento e uma lista e devolve o número de ocorrências na lista do elemento fornecido. Por exemplo,

```
> (conta 3 ’(2 4 5))
0
> (conta 3 ’(4 3 5 3 6 7 3 5))
3
```

17. Defina um procedimento `junta-listas-ordenadas`, que recebe duas listas de inteiros, ordenadas por ordem crescente, e devolve uma lista também ordenada com os elementos das duas listas. Na lista devolvida não devem existir elementos repetidos. Por exemplo,

```
> (junta-listas-ordenadas (list 1 5 6) (list 2 4 6 9))
(1 2 4 5 6 9)
```

18. Escreva um procedimento chamado `membro?` que recebe um inteiro e uma lista de inteiros e devolve *verdadeiro* apenas se o elemento existe na lista de inteiros.

19. Escreva um procedimento `remove` que recebe um elemento e uma lista e devolve a lista após a remoção de todas as ocorrências do elemento fornecido. Sugestão: utilize o procedimento `membro?` da pergunta anterior. Por exemplo,

```
> (remove 3 '(2 4 3 3 5 6 3 8 8))
(2 4 5 6 8 8)
> (remove 2 '(3 4 5))
(3 4 5)
```

20. Escreva um procedimento `elementos-repetidos` que recebe uma lista de inteiros e devolve uma lista com os elementos que aparecem repetidos nessa lista. A ordem pela qual os elementos aparecem na lista devolvida é irrelevante. Sugestão: utilize os procedimentos `membro?` e `remove` das perguntas anteriores.

    Por exemplo,

```
> (elementos-repetidos '(1 2 3 2 3 3))
(3 2)
```

21. Escreva um procedimento chamado `unicos` que remove elementos duplicados de uma lista. Por exemplo,

```
> (unicos '(1 1 2 3 4 2 4 5 5 5 5 1 6 5))
(1 2 3 4 5 6)
```

22. Utilizando as funcionais sobre listas escreva um procedimento que recebe uma lista de inteiros e que devolve a soma dos quadrados dos elementos da lista.

23. Escreva um procedimento chamado `conta-elementos` que recebe uma lista de inteiros e que devolve uma lista de pares em que cada par contém como primeiro elemento um elemento da lista e como segundo elemento o número de vezes que este aparece na lista. A ordem pela qual os pares aparecem é irrelevante.

    Respeite as barreiras de abstracção, mas lembre-se que está a lidar com dois tipos estruturados, a lista e o par. Por exemplo,

    ```
    > (conta-elementos ‘(3 4 5 5 3 3 9 3 3 12 3 12))
    ((3 . 6)(4 . 1)(5 . 2)(9 . 1)(12 .2))
    ```

24. Escreva um procedimento, `faz-troco`, que recebe um número real correspondente a uma quantia em euros e a lista dos valores das notas e moedas de euro e produz uma lista que indica as notas e moedas necessárias para perfazer a quantia, usando o maior número possível de notas e moedas de valor mais elevado.

25. Escreva um procedimento chamado `alisa` que recebe uma lista, cujos elementos podem ser listas, e devolve uma lista com todos os elementos atómicos da lista original. Por exemplo,

    ```
    > (alisa ‘((((3))) 4 5 (8 (9))))
    (3 4 5 8 9)
    ```

26. Escreva um procedimento chamado `parte` que recebe um inteiro e uma lista de inteiros e que devolve uma lista contendo duas listas, verificando as duas seguintes condições: (1) todos os elementos da primeira lista são menores que o inteiro recebido e (2) todos da segunda lista são maiores que o elemento recebido. Por exemplo,

    ```
    > (parte 3 ‘(6 1 2 9 7))
    ((1 2)(6 9 7))
    > (parte 3 ‘(6 1 2 3 9 7))
    ((1 2)(6 9 7))
    > (parte 4 ‘(9 12 5))
    (() (9 12 5))
    ```

27. (a) Escreva um procedimento chamado `da-pares` que recebe um procedimento, um valor e uma lista e que devolve uma lista de pares em que cada par contém o elemento da lista seguido do valor obtido da aplicação do procedimento ao valor e ao elemento da lista, respectivo. Por exemplo:

```
> (da-pares (lambda (x y) (+ x y)) 3 '(2 4 3 5 6))
((2 5) (4 7) (3 6) (5 8) (6 9))
> (da-pares (lambda (x y) (* x y)) 2 ())
()
> (da-pares (lambda (x y) (* x y)) 9 '(1 3))
((1 9) (3 27))
```

(b) O seu procedimento gera um processo iterativo ou recursivo? Justifique a resposta, explicando o que distingue um processo iterativo de um processo recursivo.

28. A interacção

```
(define lista1 (list 1 3 4 7))
```

consititui uma quebra das barreiras de abstracção. Com efeito, uma vez que `(list el1 el2 ... eln)` é açúcar sintáctico para

```
(cons el1
      (cons el2
            (cons ...
                  ...
                    (cons eln ()))))
```

quando usamos o procedimento primitivo `list` é como se estivéssemos a usar `cons`. Ou seja, estamos a quebrar a barreira de abstracção criada com a definição das operaçoes básicas para listas. Assim, só podemos usar `list` para criar listas, se a representação escolhida para listas for a que escolhemos neste capítulo. Escreva um procedimento `lista` que recebe uma lista (do Scheme), e devolve uma lista abstracta, isto é, construída usando os construtores para listas.

29. Defina um procedimento chamado `sublistas`, que recebe uma lista e que conta o número total de listas que esta contém. Por exemplo:

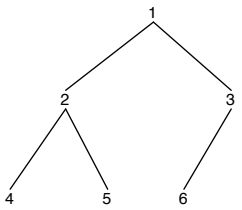
```
> (sublistas '(1 2 3))
0
> (sublistas '((1) 2 (3)))
2
> (sublistas '((((((1))))))
4
```

30. Diga o que acontece quando se avalia em Scheme cada uma das seguintes formas, justificando a diferença de comportamento do avaliador.

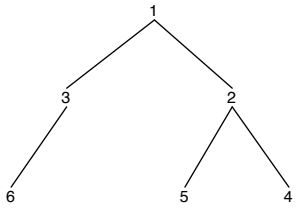
```
> ((lambda (x) (* x x)) 3)
> '((lambda (x) (* x x)) 3)
```

5.2.3 Árvores

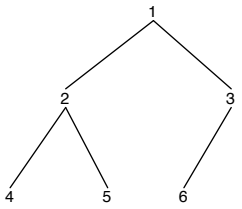
1. Defina um procedimento `espelha-arv` que recebe uma árvore binária e devolve uma nova árvore binária idêntica à recebida, mas em que em cada nível da árvore se trocou a árvore esquerda com a direita, mantendo-se a raíz. Por exemplo, se o procedimento `espelha-arv` receber a árvore



este irá produzir a árvore



2. Escreva um procedimento chamado `cria-arvore` que recebe zero ou mais argumentos e devolve uma árvore criada a partir desses argumentos. Por exemplo, `(cria-arvore 1 2 3 4 5 6)` devolve a seguinte àrvore:



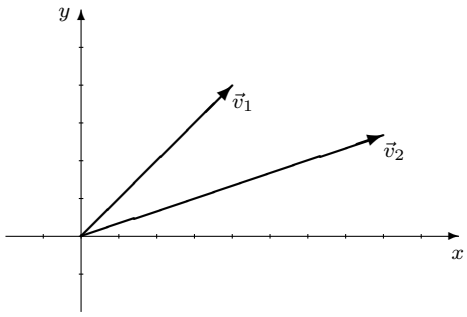


Figura 5.1: Exemplo de vectores.

3. Escreva um procedimento em Scheme que recebe uma árvore binária de procura e um elemento, e devolve a árvore binária ordenada com o elemento inserido.

4. Escreva um procedimento em scheme que recebe uma árvore binária de procura e um elemento, e devolve o valor lógico correspondente ao facto de o elemento se encontrar ou não na árvore.

5. Escreva um procedimento `procura` que recebe uma árvore binária de procura e um inteiro, e devolve *verdadeiro* se o inteiro existir na árvore, e $falso$, em caso contrário.

   Nota: O seu procedimento deve tirar partido do facto de receber uma árvore binária de procura e não uma árvore binária qualquer.

5.2.4 Definição de novos tipos

1. Defina e implemente o tipo `info-aluno`, adequado para representar a seguinte informação sobre um aluno: o número de aluno (um inteiro), o nome (uma cadeia de caracteres) e duas notas (inteiros).

2. Suponha que desejava criar o tipo vector em Scheme. Um vector num referencial cartesiano pode ser representado pelas coordenadas da sua extremidade (x, y), estando a sua origem no ponto $(0, 0)$, ver Figura 5.1.

   Podemos considerar as seguintes operações básicas para vectores:

   - *Construtor*:
     $vector : real \times real \mapsto vector$

$vector(x, y)$ tem como valor o vector cuja extremidade é o ponto (x, y).

- *Selectores*:
  $abcissa : vector \mapsto real$
  $abcissa(v)$ tem como valor a abcissa da extremidade do vector v.
  $ordenada : vector \mapsto real$
  $ordenada(v)$ tem como valor a ordenada da extremidade do vector v.
- *Reconhecedores*:
  $vector? : universal \mapsto lógico$
  $vector?(arg)$ tem valor $verdadeiro$ apenas se arg é um vector.
  $vector\_nulo? : vector \mapsto lógico$
  $vector\_nulo?(v)$ tem valor $verdadeiro$ apenas se v é o vector (0, 0).
- *Teste*:
  $vectores =? : vector \times vector \mapsto lógico$
  $vectores =?(v_1, v_2)$ tem valor verdadeiro apenas se os vectores v_1 e v_2 são iguais.

(a) Defina uma representação para vectores utilizando pares.

(b) Escreva em Scheme as operações básicas com base na representação escolhida.

(c) Escolha uma representação externa para vectores e escreva o transformador de saída.

3. Tendo em atenção as operações básicas sobre vectores da pergunta anterior, escreva procedimentos em Scheme para:

   (a) Somar dois vectores. A soma dos vectores representados pelos pontos (a, b) e (c, d) é dada pelo vector $(a + c, b + d)$.

   (b) Calcular o produto escalar de dois vectores. O produto escalar dos vectores representados pelos pontos (a, b) e (c, d) é dado pelo real $a.c + b.d$.

   (c) Determinar se um vector é colinear com o eixo dos xx.

4. Ao responder às perguntas anteriores foram criadas duas camadas de abstracção, designadas por Camada 1 e Camada 2 na figura 5.2, a qual também mostra as várias barreiras de abstracção.

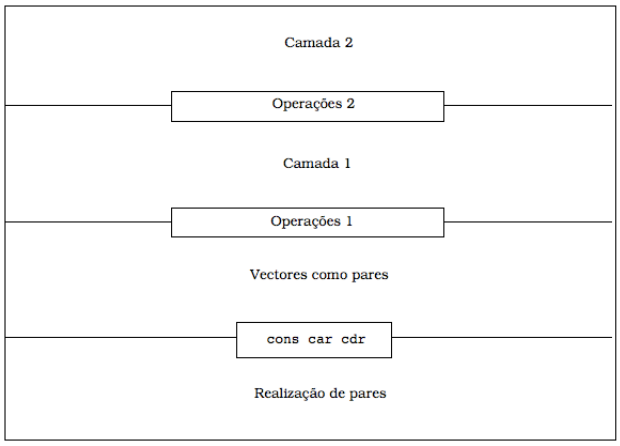


Figura 5.2: Camadas de abstracção.

(a) Diga o que existe conceptualmente em cada uma destas camadas.

(b) Das operações que foram apresentadas nas questões anteriores, diga quais pertencem a Operações 1 e quais pertencem a Operações 2.

(c) Explique a razão porque às Operações 2 não é permitido a utilização das primitivas `car`, `cdr` e `cons`.

5. Suponha que pretende criar o tipo abstracto matrícula, correspondente a matrículas portuguesas de automóveis. Uma matrícula tem a forma DD-LL-DD, em que DD são dois dígitos e LL são duas letras maiúsculas.

(a) Utilizando a abordagem dos tipos abstractos de informação, defina as operações básicas para o tipo matrícula.

Nota: assuma que as letras pertencem ao tipo cadeia de caracteres.

(b) Escolha uma representação para o tipo matrícula.

(c) Escreva em Scheme as operações básicas para o tipo matrícula tendo em conta a representação escolhida. Parta do princípio que existe o predicado `duasletras?` que recebe uma cadeia de caracteres e tem o valor verdadeiro apenas se esta for constituída por duas letras maiúsculas.

(d) Partindo do princípio que a ordem por que são geradas as matrículas é:
00-AA-00
00-AA-01
...
99-AA-99
00-AB-00
...
99-ZZ-99
escreva um procedimento em Scheme que recebe duas matrículas e que tem o valor verdadeiro apenas se a matrícula correspondente ao primeiro argumento é mais antiga do que a matrícula correspondente ao segundo argumento.

Parta do princípio que existe o predicado `>letras?` que recebe duas cadeia de caracteres, de duas letras cada, e tem o valor verdadeiro apenas se o primeiro argumento aparecer cronologicamente numa matrícula após o segundo argumento.

6. Suponha que desejava criar o tipo tempo em Scheme. Suponha que o tempo é caracterizado por um certo número de horas (um inteiro não negativo), minutos (um inteiro entre 0 e 59) e segundos (um inteiro entre 0 e 59).

   (a) Especifique as operações básicas para o tipo tempo.

   (b) Escolha uma representação interna para o tipo tempo.

   (c) Escreva em Scheme as operações básicas, de acordo com a representação escolhida.

   (d) Supondo que a representação externa para o tempo é h:m:s (em que h representa as horas, m os minutos e s os segundos) escreva o transformador de saída para o tipo tempo.

   (e) Escreva um procedimento em Scheme que calcula o número de segundos entre dois instantes de tempo. Por exemplo, o número de segundos entre os tempos 2:20:34 e 2:30:32 é de 598. O seu procedimento apenas deve produzir um valor se o segundo tempo for maior do que o primeiro, gerando uma mensagem de erro se essa condição não se verificar.

7. Suponha que desejava criar o tipo data em Scheme. Suponha que a data é caracterizada por um dia (um inteiro entre 1 e 31), um mês (um inteiro entre 1 e 12) e um ano.

(a) Especifique as operações básicas para o tipo data.

(b) Escolha uma representação interna para o tipo data.

(c) Escreva em Scheme as operações básicas, de acordo com a representação escolhida.

(d) Supondo que a representação externa para a data é $d/m/a$ (em que d representa o dia, m o mês e a o ano) escreva o transformador de saída para o tipo data.

(e) Tendo em conta as operações básicas escreva um procedimento em Scheme que permite adicionar uma semana a uma determinada data.

8. O tipo de informação *pilha* corresponde a uma pilha de objectos físicos, à qual podemos apenas aceder ao elemento no topo da pilha e apenas podemos adicionar elementos ao topo da pilha.

   O tipo pilha é caracterizado pelas operações: *nova_pilha* (cria uma pilha sem elementos), *empurra* (adiciona um elemento à pilha), *topo* (indica o elemento no topo da pilha), *tira* (remove o elemento no topo da pilha), *pilha*? (decide se um objecto computacional é uma pilha), *pilha_vazia*? (testa a pilha sem elementos) e *pilhas*=? (testa a igualdade de pilhas).

   (a) Especifique formalmente estas operacões, e classifique-as em construtores, selectores, reconhecedores e testes.

   (b) Escolha uma representação para o tipo pilha, e com base nesta, escreva as operações básicas.

9. Suponha que desejava criar o tipo conjunto em Scheme. Considere as seguintes operações para conjuntos:

   Construtores:

   - $novo\_conj : \{\} \mapsto conjunto$
     $novo\_conj()$ tem como valor um conjunto sem elementos.
   - $insere\_conj : elemento \times conjunto \mapsto conjunto$
     $insere\_conj(e, c)$ tem como valor o resultado de inserir o elemento e no conjunto c; se e já pertencer a c, tem como valor c.

   Selectores:

- $retira\_conj : elemento \times conjunto \mapsto conjunto$
  $retira\_conj(e, c)$ tem como valor o resultado de retirar do conjunto c o elemento e; se e não pertencer a c, tem como valor c.

Reconhecedores:

- $conjunto? : universal \mapsto lógico$
  $conjunto?(arg)$ tem o valor $verdadeiro$ se arg é um conjunto, e tem o valor $falso$, em caso contrário.
- $conj\_vazio? : conjunto \mapsto lógico$
  $conj\_vazio?(c)$ tem o valor $verdadeiro$ se o conjunto c é o conjunto vazio, e tem o valor $falso$, em caso contrário.

Testes:

- $conjuntos{=}? : conjunto \times conjunto \mapsto lógico$
  $conjuntos{=}?(c_1, c_2)$ tem o valor $verdadeiro$ se os conjuntos c_1 e c_2 forem iguais, ou seja, se tiverem os mesmos elementos, e tem o valor $falso$, em caso contrário.
- $pertence? : elemento \times conjunto \mapsto lógico$
  $pertence?(e, c)$ tem o valor $verdadeiro$ se o elemento e pertence ao conjunto c e tem o valor $falso$, em caso contrário.

Operações adicionais:

- $cardinal : conjunto \mapsto inteiro$
  $cardinal(c)$ tem como valor o número de elementos do conjunto c.
- $subconjunto? : conjunto \times conjunto \mapsto lógico$
  $subconjunto?(c_1, c_2)$ tem o valor $verdadeiro$, se o conjunto c_1 for um subconjunto do conjunto c_2, ou seja, se todos os elementos de c_1 pertencerem a c_2, e tem o valor $falso$, em caso contrário.
- $uniao : conjunto \times conjunto \mapsto conjunto$
  $uniao(c_1, c_2)$ tem como valor o conjunto união de c_1 com c_2, ou seja, o conjunto formado por todos os elementos que pertencem a c_1 ou a c_2.
- $interseccao : conjunto \times conjunto \mapsto conjunto$
  $interseccao(c_1, c_2)$ tem como valor o conjunto intersecção de c_1 com c_2, ou seja, o conjunto formado por todos os elementos que pertencem simultaneamente a c_1 e a c_2.

- $diferenca : conjunto \times conjunto \mapsto conjunto$
  $diferenca(c_1, c_2)$ tem como valor o conjunto diferença de c_1 e c_2, ou seja, o conjunto formado por todos os elementos que pertencem a c_1 e não pertencem a c_2.

(a) Escreva uma axiomatização para as operações do tipo conjunto.

(b) Defina uma representação para conjuntos utilizando pares.

(c) Escreva em Scheme as operações básicas, com base na representação escolhida.

10. Suponha que quer representar um tipo muito simples de dicionário, que, para cada palavra numa determinada língua (chave), faz corresponder uma outra palavra noutra língua (tradução).

    Para isso foi criado o tipo abstracto de informação *dicionario*, com as seguintes operações:

    - $novo\_dicionario : \{\} \mapsto dicionario$
      Cria um novo dicionário, sem entradas nenhumas.
    - $dicionario\_insere\_entrada : dicionario \times cadeia^2 \mapsto dicionario$
      Cria um novo dicionário, idêntico ao dicionário recebido, mas com uma nova entrada, cuja chave é o segundo argumento e a tradução é o terceiro argumento. Se já existir uma entrada no dicionário original com a mesma chave, a nova entrada substitui a anterior.
    - $dicionario\_remove\_entrada : dicionario \times cadeia \mapsto dicionario$
      Devolve um dicionário idêntico ao recebido mas sem a entrada cuja chave é o segundo argumento.
    - $dicionario\_procura\_entrada : dicionario \times cadeia \mapsto cadeia ou falso$
      Recebe um dicionário e uma chave e devolve a tradução para essa chave ou *falso* se não existir nenhuma entrada com essa chave no dicionário.

    Exemplos de utilização deste tipo de dados:

```
>(define dicio
(dicionario-insere-entrada
   (dicionario-insere-entrada (novo-dicionario) "Eu""I")
   "adoro"
   "love"))
```

```
>(dicionario-procura-entrada dicio "Eu")
"I"
>(dicionario-procura-entrada
   (dicionario-remove-entrada dicio "Eu")
   "Eu")
#f
```

(a) Escolha uma representação para este tipo de dados.

(b) Implemente as operações de acordo com a representação escolhida.

11. Em relação à pergunta anterior, suponha que pretende definir o procedimento `traduz` que recebe uma lista de strings, correspondendo a uma frase a ser traduzida, um dicionário, e um procedimento de um argumento, e devolve uma lista de strings correspondendo à tradução da frase original. O procedimento recebido como argumento deve ser chamado com cada palavra que não existe no dicionário e deve devolver o valor a utilizar como tradução para essa palavra. Por exemplo, usando o nome `dicio` definido acima:

```
>(traduz '("Eu" "adoro" "Scheme")
dicio
(lambda (x) x))
("I" "love" "Scheme")

>(traduz '("Eu" "adoro" "Scheme")
dicio
(lambda (x) "??"))
("I" "love" "??")
```

(a) Este procedimento deve ou não fazer parte do tipo de dados `dicionario`? Justifique a sua resposta.

(b) Defina o procedimento.

(c) Um valor útil a passar como terceiro argumento para o procedimento `traduz` seria um procedimento que perguntasse ao utilizador qual a tradução para as palavras não conhecidas no dicionário.

Vamos supor que esse procedimento existe e que se chama `pede-traducao`. No entanto, nos casos em que a mesma palavra desconhecida ocorre mais do que uma vez, não devia ser pedida a tradução dessa palavra mais do que uma vez ao utilizador:

```
>(traduz '("Eu" "gosto" "de" "Scheme" "e" "de" "Lisp")
  dicio
  pede-traducao)
Traducao para "gosto"? "love"
Traducao para "de"? ""
Traducao para "Scheme"? "Scheme"
Traducao para "e"? "and"
Traducao para "de"? ""
Traducao para "Lisp"? "Lisp"
("I" "love" "" "Scheme" "and" "" "Lisp")
```

Repare que no exemplo acima, foi pedida duas vezes a tradução da palavra `de`. Defina o procedimento `cria-tradutor` que devolve um procedimento com estado interno que pede a tradução das palavras ao utilizador mas vai construindo um dicionário interno com as traduções já dadas de forma a não pedir duas vezes a mesma tradução. Ou seja,

```
>(traduz '("Eu" "gosto" "de" "Scheme" "e" "de" "Lisp")
  dicio
  (cria-tradutor))
Traducao para "gosto"? "love"
Traducao para "de"? ""
Traducao para "Scheme"? "Scheme"
Traducao para "e"? "and"
Traducao para "Lisp"? "Lisp"
("I" "love" "" "Scheme" "and" "" "Lisp")
```

12. Considere um tipo de dados *sequência* que consiste num conjunto de inteiros, cujos elementos estão ordenados e em que não podemos ter repetições de valores. O número de elementos da sequência é chamado o seu comprimento. Os elementos de uma sequência podem ser referenciados por inteiros entre 0 e comprimento - 1.

    (a) Especifique e implemente para este tipo de dados, os construtores (cria-sequencia, insere-elemento), os selectores (elemento-posicao, comprimento-sequencia), o reconhecedor (sequencia-vazia?) e o teste (sequencias-iguais?)

    (b) Com base no tipo implementado, defina a operação que dada uma sequência devolve a diferencia entre o maior e o menor elementos dessa sequencia.

13. Suponha que desejava criar o tipo turma em Scheme. Suponha que a turma é caracterizada por um nome, que corresponde ao identificador da turma, por uma lista dos números dos alunos da turma (ordenada por ordem crescente), e por uma representação dos limites, que corresponde ao número de aluno mais baixo, e o mais alto pertencentes a essa turma.

    (a) Especifique as operações básicas para o tipo turma suficientes para: criar uma nova turma a partir do identificador da turma e dos números dos alunos pertencentes à turma; seleccionar os vários componentes de uma turma (nome, alunos e limites); inserir um novo aluno numa turma; remover um aluno de uma turma; saber se um aluno pertence a uma turma; e saber o número de alunos de uma turma.

    (b) Escolha uma representação interna para o tipo turma.

    (c) Escreva em Scheme as operações básicas do tipo turma, de acordo com a representação escolhida. Nota: pode usar o procedimento ordena que recebe uma lista e devolve a mesma lista ordenada.

    Exemplo de interacção:

```
> (define t1 (cria-turma  'T123   '( 54232  53213 56789)))
> (mostra t1)
TURMA: T123
ALUNOS: 53213  54232  56789
LIMITES: 53213  -  56789
> (insere-aluno t1 53213)
"Aluno já existe"
> (insere-aluno t1 58231)
> (mostra t1)
TURMA: T123
ALUNOS: 53213  54232  56789 58231
LIMITES: 53213  -  58231
```

Capítulo 6

O desenvolvimento de programas

6.1 Exercícios de revisão

1. Diga quais as fases por que passa o desenvolvimento de um programa e o que se faz em cada uma delas.

2. Enuncie e explique cada um dos tópicos que são habitualmente focados na documentação de utilização que acompanha um programa.

3. Em programação podemos ter erros sintácticos e erros semânticos. Defina cada um deles e dê exemplos.

4. O que é a depuração? Que tipos de depuração são efectuados nos programas e para que tipos de erros?

5. O que é a depuração da base para o topo? Qual a sua vantagem?

6. Comente a seguinte expressão: "A documentação de um programa deve ser efectuada após o código terminado, para garantir que não existem erros nessa mesma documentação".

6.2 Exercícios de programação

1. Durante o desenvolvimento de um programa, após a programação da solução, segue-se a fase de testes.

   (a) Explique no que consiste a fase de testes.

(b) Concretize, sugerindo casos de teste para o procedimento que se segue:

```
(define (saudacao hora)
   (cond ((> hora 12) "Boa tarde !")
         ((> hora 19) "Boa noite !")
         (else "Bom dia !")))
```

(c) Com base nos casos de teste concebidos explique qual o problema com o procedimento anterior.

2. Durante o desenvolvimento de um programa, após a programação da solução, segue-se a fase de testes.

(a) Explique no que consiste a fase de testes.

(b) Concretize a resposta à alínea anterior, sugerindo casos de teste para o procedimento seguinte:

```
(define (mistério a)
          (cond ((> a 0) #t)
                ((< a 3) #t)
                ((= a 1) (+ a 5))
                (else (mistério (- a 2)))))
```

(c) Com base nos casos de teste concebidos explique qual o problema com o procedimento anterior.

Capítulo 7

Programação imperativa

7.1 Exercícios de revisão

1. Distinga entre programação imperativa e programação funcional.
2. Explique a necessidade da introdução do operador de atribuição.
3. Qual a diferença entre o operador de atribuição e o operador de nomeação?
4. Diga quais as vantagens e os inconvenientes da programação imperativa.
5. Diga o que é um efeito. Qual o comportamento de um procedimento baseado em efeitos?
6. Quando é que se diz que um objecto tem estado? O que caracteriza o estado de um objecto?
7. Explique a diferença entre o método de passagem de parâmetros por valor e por referência. Explique estes dois métodos com exemplos em Scheme.

7.2 Exercícios de programação

7.2.1 Utilização da atribuição

1. Considere a operação potência, em que tanto a base como o expoente são números naturais. Escreva, usando programação imperativa, o procedimento `potencia` que recebe dois números naturais e devolve

o resultado de elevar o primeiro à potência especificada pelo segundo argumento.

2. Diga qual a diferença entre

```
(define b 3)
(define (p a)
  (if (= a 0)
      (set! b (sub1 b))
      (set! b a))
  (add1 b))
```

e

```
(define b 3)
(define (p a)
  (if (= a 0)
      (set! b (sub1 b))
      (begin
        (set! b a)
        (add1 b))))
```

7.2.2 Vectores

1. Escreva um procedimento em Scheme que recebe como argumentos um vector e um inteiro e devolve *verdadeiro*, se o inteiro está armazenado no vector, e $falso$, em caso contrário.

2. Defina o procedimento aplica que recebe como argumentos um vector e uma função, e devolve o resultado de aplicar essa função à soma dos elementos do vector.

3. Defina o procedimento `maiores-elementos` que recebe como argumentos um vector e um elemento, e produz um novo vector com os elementos do vector original que sejam maiores que o elemento recebido como argumento.

4. Crie um procedimento Scheme `quadrado-vector` que recebe um vector e devolve um novo vector cujos os elementos são o quadrado dos elementos do vector recebido. Exemplos de interacção:

```
>(define v (vector 1 2 3))
>(quadrado-vector v)
#3(1 4 9)
>v
#3(1 2 3)
```

5. Escreva uma versão destrutiva `quadrado-vector!` que tem o mesmo comportamento do `quadrado-vector` mas que altera destrutivamente o vector que recebe. Exemplos de interacção:

```
>(quadrado-vector! v)
#3(1 4 9)
>v
#3(1 4 9)
```

6. Defina o procedimento `aplica-talvez` que recebe como argumentos um vector, um elemento, e uma função, e que devolva o vector alterado destrutivamente, resultado de aplicar a função aos elementos do vector que sejam iguais ao elemento recebido como argumento.

7. Defina um procedimento `vector-conta` que recebe um elemento e um vector e devolve o número de ocorrências do elemento no vector. Use o predicado `eq?` para comparar o elemento com cada um dos elementos do vector. Por exemplo:

```
> (vector-conta 'a (vector 'a 'b 'c 'a 'b 'c))
2
> (vector-conta 'a (vector))
0
```

8. Usando o procedimento `vector-conta` do exercício anterior, defina um procedimento `vector-remove` que recebe um elemento e um vector e devolve um novo vector com todos os elementos do vector original que não sejam iguais ao elemento indicado. Assuma que os elementos são comparáveis usando o predicado `eq?`. Por exemplo:

```
> (vector-remove 'a (vector 'a 'b 'c 'a 'b 'c))
#(b c b c)
> (vector-remove 'a (vector))
#()
```

9. Defina um procedimento `ocorre-par` que recebe um vector e um objecto e retorna `#t` se o vector contiver esse objecto um número par de vezes e `#f` caso contrário. Note que 0 é um número par.

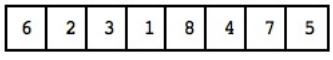
10. Defina um procedimento que recebe um vector e devolve uma lista com todos os elementos do vector. Pode utilizar as operações primitivas de listas e vectores que achar necessárias.

11. Defina um procedimento `vector-max` que recebe um vector de inteiros e devolve o maior inteiro contido no vector. Se um dos elementos do vector não for um inteiro ou se o parâmetro não for um vector, o procedimento deve imprimir um erro no ecrã e terminar a sua execução. Assuma que, se for passado um vector, este tem pelo menos um elemento.

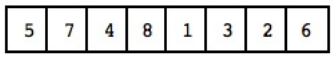
12. Escreva um procedimento de ordem superior, chamado `aplica-vector`, que recebe um vector contendo números e um procedimento de um argumento aplicável a números e que modifica o vector recebido, aplicando o procedimento a todos os elementos do vector. Por exemplo, utilizando o procedimento `escreve-vector`, podemos obter a seguinte interacção:

```
> (define v (vector 1 2 3 4 5))
> (escreve-vector v)
[1 2 3 4 5]
> (aplica-vector (lambda (x)(* x x)) v)
> (escreve-vector v)
[1 4 9 16 25]
> (aplica-vector (lambda (x)(* x x)) v)
> (escreve-vector v)
[1 16 81 256 625]
```

13. Escreva um procedimento em Scheme que recebe como argumento um vector e que altera esse vector trocando os elementos nas posições simétricas em relação ao elemento que se encontra no centro do vector. Por exemplo, se o vector for



o vector devolvido será:



14. Defina um procedimento `procura` que recebe um número e um vector e devolve a posição no vector onde esse número ocorre, ou o valor lógico `#f` no caso de o número não existir no vector. Por exemplo,

```
> (define v1 (vector 7 2 3 4 5 1))
> (procura v1 4)
3
> (procura v1 9)
#f
```

15. Uma *matriz* é uma tabela bidimensional em que os seus elementos são referenciados pela linha e pela coluna em que se encontram. Defina as operações básicas para o tipo matriz. Realize as operações básicas com base em vectores. Nota: uma matriz pode ser considerada como um vector cujos elementos são vectores.

16. Considere o tipo matriz definido no exercício anterior. Escreva um procedimento que corresponde ao transformador de saída para matrizes, escrevendo uma matriz sob a forma

$$\begin{array}{ccccc} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \cdots & \cdots & & & \cdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{array}$$

17. Considere, de novo, o tipo matriz. Escreva um procedimento em Scheme que recebe como argumentos duas matrizes e devolve uma matriz correspondente ao produto das matrizes que são seus argumentos. Os elementos da matriz produto são dados por

$$p_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

7.2.3 Procura e ordenação

1. Um método de ordenação muito eficiente, chamado "*quick sort*", consiste em considerar um dos elementos a ordenar (em geral, o primeiro elemento do vector), e dividir os restantes em dois grupos, um deles

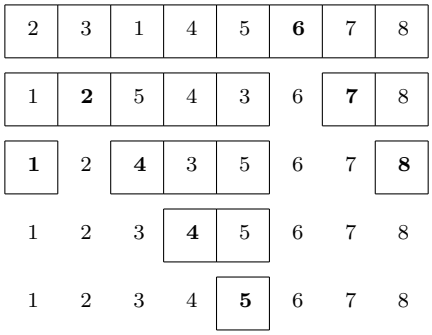
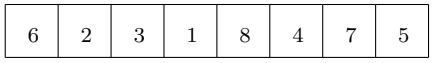


Figura 7.1: Passos seguidos no "*quick sort*".

com os elementos menores e o outro contém os elementos maiores que o elemento considerado. Este é colocado entre os dois grupos, que são por sua vez ordenados utilizando "*quick sort*". Por exemplo, os passos apresentados na Figura 7.1 correspondem à ordenação do vector



utilizando "*quick sort*"(o elemento escolhido em cada vector representa-se a carregado). Escreva um procedimento em Scheme para efectuar a ordenação de um vector utilizando o "*quick sort*".

7.2.4 Procedimentos com estado interno

1. Escreva um procedimento com estado interno, chamado `conta-acessos`, que recebe um símbolo correspondente a uma *password* e que indica o número de vezes que o procedimento foi invocado com a *password* certa e com a *password* errada. O procedimento é criado indicando a *password*. Por exemplo,

```
> (define acessos-1 (conta-acessos 'xpto))
> (acessos-1 'xpto)
invocações válidas: 1
invocações inválidas: 0
> (acessos-1 'xx)
```

```
invocações válidas: 1
invocações inválidas: 1
> (acessos-1 'xpto)
invocações válidas: 2
invocações inválidas: 1
```

2. O seguinte procedimento para calcular os números de Fibonacci usando um processo recursivo não é eficiente, porque calcula múltiplas vezes o mesmo valor.

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```

   Para evitar a repetição de cálculos, podemos imaginar um procedimento com estado interno que vai memorizando os números de Fibonacci, à medida que estes são calculados. Para isso, este procedimento mantém uma lista com os valores dos números de Fibonacci que já memorizou, por exemplo, `((2 . 1) (1 . 1) (0 . 0))`, e, sempre que tem de calcular um número de Fibonacci, primeiro vai ver se já sabe o seu valor e, se não o souber, calcula o seu valor e memoriza-o.

   Escreva um procedimento em Scheme para calcular os números de Fibonacci, de acordo com esta técnica.

3. Escreva um procedimento `cria-proc-historial` que recebe um procedimento de um argumento e devolve um procedimento que mantém uma lista com o historial dos argumentos aos quais foi aplicado. O primeiro elemento do historial deve ser o último argumento a ser utilizado na função e o o último elemento do historial deve ser o primeiro argumento ao qual foi aplicado a função. Para aceder ao historial passa-se o simbolo historial à função. Exemplos de interacção:

```
>(define quadrado-historial
    (cria-proc-historial (lambda (x) (* x x))
>(quadrado-historial 1)
1
>(quadrado-historial 2)
4
```

```
>(quadrado-historial 3)
9
>(quadrado-historial 'historial)
(3 2 1)
```

4. Pretende-se simular um jogo cujo objectivo é acertar num numero, "pensado"pelo computador, entre 0 e 10. Quando se acerta é devolvido o número de tentativas que foram necessárias, e é pensado um novo número.

```
>(define meu-jogo (acerta-num))
>(meu-jogo 3)
Nao acertou. Tente de novo...
>(meu-jogo 5)
Nao acertou. Tente de novo...
>(meu-jogo 0)
3
>(meu-jogo 0)
Nao acertou. Tente de novo...
```

   Sugestão: Use o procedimento `random`, já definido em Scheme, que gera numeros aleatórios `(random n)` = inteiro entre 0 e n.

5. O seguinte procedimento para calcular os números de Fibonacci usando um processo recursivo não é eficiente, porque calcula múltiplas vezes o mesmo valor.

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```

   Para evitar a repetição de cálculos, podemos imaginar um procedimento com estado interno que vai memorizando os números de Fibonacci, à medida que estes são calculados. Para isso, este procedimento mantém uma lista com os valores dos números de Fibonacci que já memorizou, por exemplo, `((0 0) (1 1) (2 1))`, e, sempre que tem de calcular um número de Fibonacci, primeiro vai ver se já sabe o seu valor e, se não o souber, calcula o seu valor e memoriza-o.

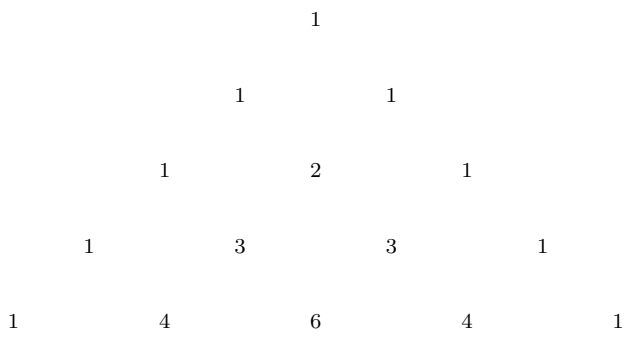


Figura 7.2: Triângulo de Pascal com 5 linhas.

Escreva um procedimento em Scheme para calcular os números de Fibonacci, de acordo com esta técnica. Recorde que os números de Fibonacci são dados por

$$fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{se } n > 1 \end{cases}$$

6. O número de combinações de m objectos n a n pode ser dado pela seguinte fórmula:

$$C(m,n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } m = n \\ C(m-1,n) + C(m-1,n-1) & \text{se } m > n,\ m > 0 \text{ e } n > 0 \end{cases}$$

Escreva um procedimento em Scheme para calcular o número de combinações de m objectos n a n e que origina um processo iterativo.

Sugestão: Baseie-se no cálculo de combinações de m objectos n a n usando o triângulo de Pascal. O triângulo de Pascal (Figura 7.2) é constituído por $m + 1$ linhas (numeradas de 0 a m), sendo a linha l constituída por $l + 1$ inteiros (colunas 0 a l). O valor de $C(m, n)$ é dado pelo valor da linha m, coluna n.

O preenchimento da linha m do triângulo é feito da seguinte forma (ver Figura 7.3):

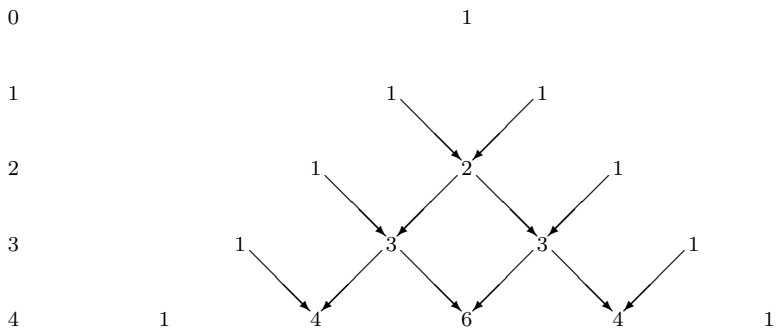


Figura 7.3: Cálculo dos elementos do triângulo de Pascal.

- O primeiro e o último elementos são ambos iguais a 1. Note-se que estes valores são determinados pelos dois primeiros casos da definição apresentada: $n = 0$ ou $m = n$.
- O elemento da coluna n da linha m é a soma dos elementos da linha $m - 1$, das colunas n e $n - 1$. Note-se que este valor é determinado pelo terceiro caso da definição apresentada: $m > n$, $m > 0$ e $n > 0$.

7. Considere o tipo `info-aluno` do Exercício 1 da Secção 5.2.4. Defina um procedimento, `ordena`, que recebe como argumentos um símbolo (que pode ser `numero` ou `nome`) e um vector de elementos do tipo `info-aluno`. O seu procedimento deve devolver um vector de índices contendo uma indicação da ordem pela qual os elementos do vector original devem aparecer para se encontrarem ordenados por número ou nome, conforme o primeiro argumento seja `numero` ou `nome`, respectivamente. Por exemplo, se `cria-aluno` for o construtor do tipo `info-aluno`, teríamos a seguinte interacção:

```
> (define al1 (cria-aluno 54434 "António Pontes" 12 14))
> (define al2 (cria-aluno 50434 "Zacarias Calado" 15 18))
> (define al3 (cria-aluno 54034 "Carlos Silva" 13 14))
> (define al4 (cria-aluno 40434 "Manuel Fontes" 15 17))
> (define alunos (vector al1 al2 al3 al4))
> (define indices-por-numero (ordena 'numero alunos))
> (escreve-vector indices-por-numero)
 [3 1 2 0]
```

```
> (define indices-por-nome (ordena 'nome alunos))
> (escreve-vector indices-por-nome)
[0 2 3 1]
```

8. Escreva um procedimento, `escreve-alunos`, que recebe um vector de alunos e um vector de índices e escreve os elementos do vector de alunos pela ordem indicada no vector de índices. Por exemplo, considerando a interacção mostrada no exercício anterior, obter-se-ia:

```
> (escreve-alunos alunos indices-por-numero)
40434 Manuel Fontes 15 17
50434 Zacarias Calado 15 18
54034 Carlos Silva 13 14
54434 António Pontes 12 14
> (escreve-alunos alunos indices-por-nome)
54434 António Pontes 12 14
54034 Carlos Silva 13 14
40434 Manuel Fontes 15 17
50434 Zacarias Calado 15 18
```

9. Escreva um procedimento, `procura`, que recebe um vector de alunos e um número ou um nome, e efectua a procura do aluno correspondente no vector alunos. O seu procedimento deve utilizar procura binária. Considere que o seu procedimento tem acesso aos vectores `indices-por-numero` e `indices-por-nome` usados no exercício anterior. Por exemplo, continuando a usar o mesmo vector `alunos`, teríamos

```
> (procura alunos 40434)
3
> (procura alunos "Zacarias Calado")
1
> (procura alunos 49434)
-1
```

Capítulo 8

Avaliação baseada em ambientes

8.1 Exercícios de revisão

1. Em que condições se diz que uma variável está não ligada ("*unbound*") num dado ambiente?

2. Como se define o ambiente associado a um enquadramento?

3. O que é um enquadramento? Qual a restrição que lhe é imposta?

4. No modelo baseado em ambientes, como se define o valor de uma variável?

5. O que é o ambiente global? Que nomes é que este contém?

6. O que é um ambiente local? Quando é que este "desaparece"?

7. Considere o diagrama de ambientes apresentado na Figura 8.1.

   (a) Quantos ambientes identifica? Justifique a sua resposta.

   (b) Qual o valor da variável a no ambiente definido pelo enquadramento E2? Justifique a sua resposta.

8. O que é um ambiente local? Quando é que este "desaparece"?

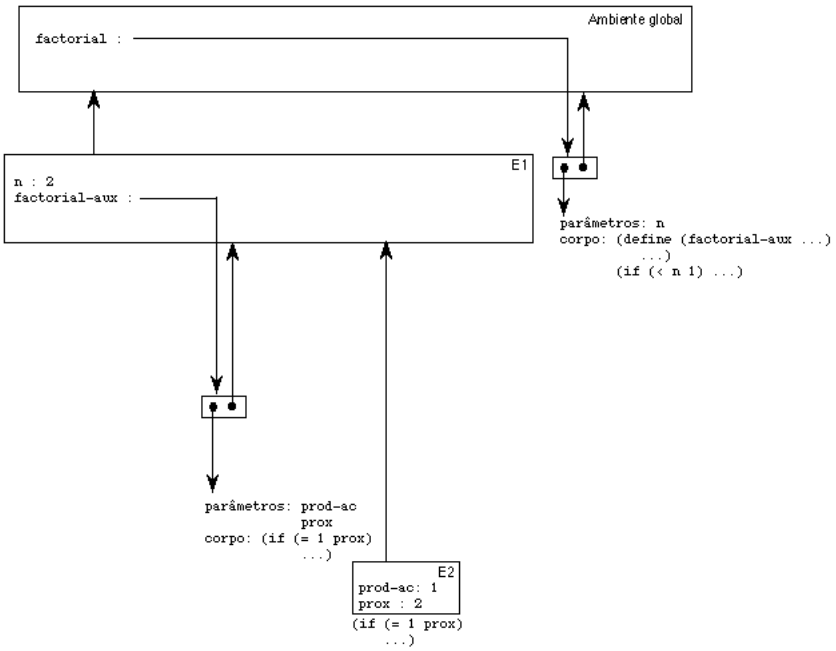


Figura 8.1: Ambientes.

8.2 Exercícios de programação

1. Desenhe o diagrama de ambientes criado pelas seguintes avaliações:

```
> (define (conjuncao p1 p2)
    (lambda (x) (and (p1 x) (p2 x))))
> (define par? even?)
> (define par>3? (conjuncao par? (lambda (x) (> x 3))))
> (par>3? 4)
#t
> (par>3? 2)
#f
> (par>3? 5)
#f
```

2. Condidere a seguinte interacção em Scheme:

```
> (define (teste)
    (define a 5)
    (define b 12)
    (+ a b))
> (teste)
17
> a
Error reference to undefined identifier: a
```

Usando um diagrama de ambientes, explique o resultado desta interacção.

3. Para cada uma das seguintes interacções, desenhe o diagrama de ambientes criados e mostre de forma clara o resultado final produzido.

(a)
```
> (define (procA1 x y)
    (define (procA2 x)
      (define (procA3 y)
        (+ x y))
      (procA3 x))
    (procA2 y))
> (procA1 2 3)
```

(b)
```
> (define (procB1 x y)
    (define (procB2 x)
      (procB3 x))
    (define (procB3 y)
      (+ x y))
    (procB2 y))
> (procB1 2 3)
```

4. Desenhe o diagrama de ambientes criado pelas seguintes avaliações:

```
> (define (cria-acumulador valor)
    (lambda (increm)
      (set! valor (+ valor increm))
      valor))
> (define ac1 (cria-acumulador 0))
> (ac1 5)
5
> (ac1 5)
10
```

5. Desenhe o diagrama de ambientes para a seguinte interacção:

```
> (define (compoe f g)
    (lambda (arg)
      (f (g arg))))
> (define (complemento pred)
    (compoe not pred))
> (define impar? (complemento even?))
> (impar? 5)
```

6. Desenhe o diagrama de ambientes para a seguinte interacção:

```
> (define x 10)
> (define y
    (let ((x 2)
          (y (+ x 10)))
       (+ (let ((z 2))
          (* z z z))
         x
         y)))
```

7. Considere a seguinte definição:

```
(define (misterio x)
  (define (misterio-aux y)
    (if (= y 0)
        1
        (+ (misterio (- x 1))
           (misterio-aux (- y 1)))))
  (if (= x 0)
      2
      (misterio-aux x)))
```

   Diga, justificando através de diagramas de ambientes, qual o resultado da avaliação da expressão `(misterio 2)`.

8. Desenhe o diagrama de ambientes criado pela seguinte interacção:

```
(define (curry p x)
  (lambda (y)
    (p x y)))

(define add5 (curry + 1))
(add5 10)
```

9. Desenhe o diagrama de ambientes criado pela seguinte interacção:

```
(define (cria-conta saldo)
  (lambda (x)
    (set! saldo (+ saldo x))
    saldo))

(define (deposita c q)
  (c q))

(define (levanta c q)
  (c (- q)))

(define c1 (cria-conta 100))
(deposita c1 20)
(levanta c1 30)
```

10. Considere a seguinte definição:

```
(define (f n)
  (define (g n) (* 2 (f n)))
  (define (f m) (* m m))
  (if (zero? n)
    0
    (if (odd? n)
      (+ (g n) (f (- n 1)))
      (+ n (f (- n 1))))))
```

    Qual o valor de `(f 3)`? Justifique apresentando os diagramas de ambientes criados durante a avaliação.

11. Desenhe o diagrama de ambientes criado pela seguinte interacão:

```
(define (cria-garrafa-agua nivel-inicial)
  (lambda (valor)
    (set! nivel-inicial (- nivel-inicial valor))
    nivel-inicial))

(define agua-50cl (cria-garrafa-agua 50))
(define agua-33cl (cria-garrafa-agua 33))

(agua-50cl 20)
(agua-50cl 10)
(agua-33cl 33)
```

12. Desenhe o diagrama de ambientes gerado pela seguinte avaliação:

```
> ((lambda (x) ((lambda (y) (set! y (* x y)) y) 4)) 5)
20
```

13. Desenhe o diagrama de ambientes criado pelos seguintes exemplos de código.

```
> (define (factorial x)
    (define (factorial-aux n)
      (if (= n 1)
          1
          (* n (factorial-aux (- n 1)))))
    (if (< x 1)
        "factorial: argumento inferior a 1"
        (factorial-aux x)))
> (factorial 2)
2
```

Capítulo 9

Estruturas mutáveis

9.1 Exercícios de revisão

1. Quando se diz que uma estrutura de informação é mutável?

2. Quais os modificadores para pares que existem em Scheme? Qual a necessidade destes modificadores?

3. Diga o que é um ponteiro. Qual a característica que distingue um ponteiro dos outros tipos de informação?

4. Distinga o comportamento FIFO ("first in first out") do comportamento LIFO ("last in first out"). Quais os tipos de informação que correspondem a cada um destes comportamentos?

5. Diga quais as características das pilhas como tipo de informação. O que as distingue das listas?

6. Diga quais as características das filas como tipo de informação. O que as distingue das listas?

7. Quando se diz que um objecto computacional corresponde a lixo? Explique a necessidade do mecanismo de recolha de lixo.

8. Diga para que serve o processo de recolha de lixo ("garbage collection") e descreva os passos envolvidos.

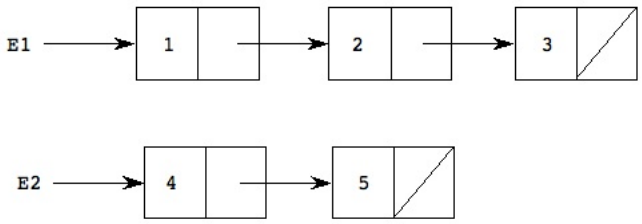
9.2 Exercícios de programação

1. (a) Represente as estruturas resultantes da seguinte interacção em Scheme:

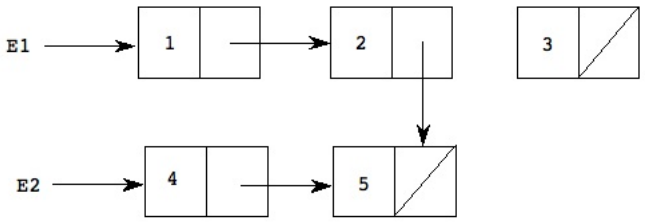
```
> (define estr1 (cons 1 (cons 2 (cons 3 ()))))
> (define estr2 (cons 1 (cons 2 ())))

> (set-cdr! estr2 (cons 5 (cdr estr1)))

> (set-cdr! estr1 5)
```

   (b) **(1.0)** Durante a avaliação das suas formas houve a criação de lixo? Justifique a sua resposta.

2. Considere as seguintes estruturas:



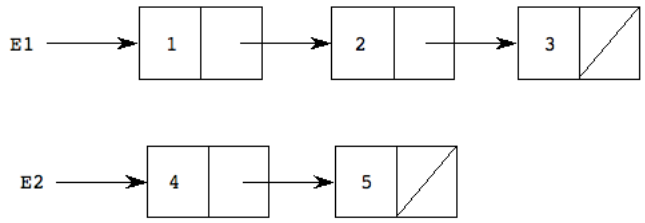
   Escreva formas em Scheme para gerar as seguintes alterações. Durante a avaliação das suas formas houve a criação de lixo? Justifique a sua resposta.



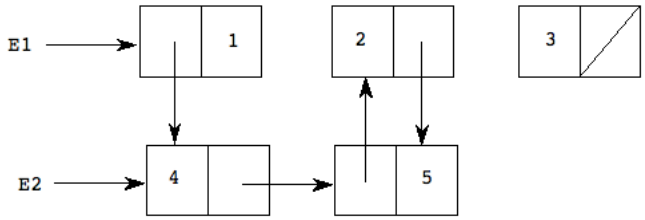
3. Represente as estruturas resultantes da seguinte interacção em Scheme:

```
> (define estr1 (cons 1 (cons 2 ())))
> (define estr2 (cons 1 (cons 2 ())))
> (define estr1 (cons 5 (cdr estr2)))
> (set-car! estr2 5)
```

4. Considere as seguintes estruturas:



Escreva formas em Scheme para gerar as seguintes alterações. Durante a avaliação das suas formas houve a criação de lixo? Justifique a sua resposta.



5. Desenhe o diagrama de caixas e ponteiros criado pela seguinte interacção. Para cada passo represente todas os nomes (`a`, `b` e `c`)

```
(define a (cons 2 3))
(define b (cons #t 6))
(define c (cons a b))
(set-car! a 20)
(set! a 20)
(set! b (list 30 74))
```

6. Desenhe o diagrama de caixas e ponteiros criado pela seguinte interacção. Para cada passo represente todos os nomes (`v`, `a` e `b`)

```
(define v (make-vector 3 0))
(define a (cons 1 2))
(define b (cons #t 2))
(vector-set! v 0 (list 9 8 7))
(vector-set! v 1 a)
```

```
(vector-set! v 2 b)
(set-car! b #f)
(set-car! a v)
```

7. Desenhe o diagrama de ambientes (com caixas ponteiros) criado pela seguinte interacção. Se ocorrer algum erro assinale-o e justifique porque este ocorreu.

```
>(define x (vector 1 2 3))
>(define aplica
   (lambda (proc x)
     (proc x)))
 (let ((x 63)
       (y x))
   (define (quadrado x)
     (* x x))
   (set! y (list 4 5 6))
   (aplica quadrado (car y)))
```

8. Desenhe o diagrama de caixas e ponteiros resultante da seguinte interacção.

```
>(define x (cons 1 (cons 2 (cons 3 ()))))
>(define y (cdr x))
>(set-cdr! x x)
>(set-car! y (car (cdr (cdr x))))
```

9. Desenhe o diagrama de caixas e ponteiros que é originado pela avaliação da seguinte forma em Scheme.

```
(let ((x (cons 1 1)))
  (set-car! x x)
  (set-cdr! x x)
  x)
```

10. Desenhe o diagrama de caixas e ponteiros que é originado pela seguinte interacção em Scheme.

```
> (define (ultimo-par x)
    (if (pair? (cdr x))
```

```
        (ultimo-par (cdr x))
        x))
> (let ((x (list 'a 'b 'c)))
    (set-cdr! (ultimo-par x) (cdr x))
    x)
```

11. Desenhe o diagrama de caixas e ponteiros que é originado pela avaliação da seguinte forma em Scheme.

```
(let ((x (list 1 2 3)))
  (let ((y (list 4 5 6)))
    (let ((z (cons x y)))
      (set-cdr! x y)
      z)))
```

12. Escreva o procedimento `junta!`, que recebe como argumentos duas listas e altera destrutivamente a primeira lista para conter os elementos da primeira lista seguidos dos elementos da segunda lista. O procedimento `junta!` não pode ser chamado com a lista vazia como primeiro argumento.

13. Escreva um procedimento `capicua` que recebe uma lista e que a altera destrutivamente construindo uma capicua com a lista original. Por exemplo:

```
>(define lt '(3 4 7 1))
>(capicua lt)
> lt
(3 4 7 1 1 7 4 3)
```

14. (a) Represente as estruturas resultantes da seguinte interacção em Scheme:

```
> (define estr1 (cons 1 (cons 2 (cons 3 ()))))
> (define estr2 (cons 1 (cons 2 ())))
> (set-cdr! estr2 (cons 5 (cdr estr1)))
> (set-cdr! estr1 5)
```

    (b) Durante a avaliação das suas formas houve a criação de lixo? Justifique a sua resposta.

Figura 9.1: Estrutura circular (caso 1).

Figura 9.2: Estrutura circular (caso 2).

15. Defina um procedimento `cria-lista-circular!` que recebe uma lista não vazia e modifica a lista de forma a que esta fique uma lista circular, ou seja, que a seguir ao último elemento da lista venha novamente o primeiro elemento da lista. O procedimento deve devolver a lista modificada. Sugestão: defina o procedimento `ultimo-par` que, dada uma lista, devolva a lista constituída pelo seu último elemento.

16. Escreva em Scheme as formas necessárias para obter as estruturas representadas nas Figuras 9.1 e 9.2. Note que estas estruturas contêm um ponteiro que aponta para a própria estrutura. Por esta razão são chamadas *estruturas circulares*.

17. Escreva um procedimento, `estrutura-circular?`, que recebe uma estrutura de pares e devolve *verdadeiro*, se se tratar de uma estrutura circular, e *falso*, em caso contrário. Por exemplo, se `est1` e `est2` forem as estruturas do exercício anterior, podemos obter a seguinte interacção:

```
> (estrutura-circular? est1)
#t
> (estrutura-circular? est2)
#t
> (estrutura-circular? ())
#f
> (estrutura-circular? (list 1))
#f
```

18. Escreva um procedimento, `desfaz-estrutura-circular!`, que recebe uma estrutura circular e a transforma na correspondente estrutura não circular, transformando o ponteiro para a própria estrutura numa

entidade que não aponta para nada. Por exemplo, continuando a usar as estruturas `est1` e `est2`, teríamos:

```
> (desfaz-estrutura-circular! est1)
> est1
(1 5 2)
> (desfaz-estrutura-circular! est2)
> est2
(1 5 2)
```

19. O tipo de informação *lista circular* corresponde a uma lista na qual, excepto no caso de ser uma lista vazia, ao último elemento se segue o primeiro. A Figura 9.3 (a) mostra esquematicamente uma lista circular em que o primeiro elemento é 4, o segundo, 3, o terceiro, 5, o quarto, 2 e o quinto (e último) é 1. Uma lista circular tem um elemento que se designa por primeiro elemento ou elemento do início da lista. Na lista anterior, esse elemento é 4.

    Com listas circulares, podemos:

    - Inserir um elemento na lista, realizado através da operação *insere_circ*. Com esta operação, o novo elemento passa a ser o primeiro da nova lista, o primeiro elemento da lista original passa a ser o segundo da nova lista, e assim sucessivamente.
    - Inspeccionar o primeiro elemento da lista, realizado através da operação *primeiro_circ*, sem alterar a lista.
    - Retirar um elemento da lista, realizado através da operação *retira_circ*. Com esta operação, o elemento retirado é sempre o do início da lista, passando o início da nova lista a ser o segundo elemento (se este existir) da lista original.
    - Avançar o início da lista para o elemento seguinte, realizado através da operação *avanca_circ*!. Esta operação não altera os elementos da lista, apenas altera o início da lista, que passa a ser o segundo elemento da lista original, se esta tiver pelo menos dois elementos; se apenas tiver um elemento, nada se altera; se a lista circular for vazia, esta operação tem um valor indefinido.

    Na Figura 9.3 estão exemplificadas estas operações.

    (a) Especifique estas operações, e outras que considerar necessárias, e classifique-as. Defina também uma representação externa para listas circulares.

Figura 9.3: Operações sobre listas circulares.

(b) Escolha uma representação interna para o tipo abstracto lista circular, e escreva em Scheme as operações básicas e o transformador de saída.

20. Em aritmética, utilizam-se parênteses para agrupar operações. Por exemplo, escrevemos $3 \times (2 + 5)$ para indicar que a operação de adição é efectuada antes da operação de multiplicação. Em expressões mais complexas utilizam-se três tipos de parênteses, parênteses curvos, parênteses rectos e chavetas. A seguinte expressão utiliza os três tipos de parênteses

$$58 + 3 \times \{[29 - 6 \times (3 - 1)] + 52\}$$

Escreva um programa em Scheme que recebe uma expressão aritmética – utilizando as quatro operações básicas (`+`, `-`, `*` e `/`) e os três tipos de parênteses – e determina se a expressão tem os parênteses colocados correctamente. Por exemplo, as expressões $2 \times (3 + 5]$ e $2 \times (3 - 7 \times [3 - 2) + 4]$ não têm os parênteses correctamente colocados. O seu programa deve utilizar uma pilha em que são colocadas as aberturas de parênteses encontradas. Sempre que é encontrado um fechar parênteses, retira-se o elemento da pilha e verifica-se se o tipo do parêntesis corresponde ao tipo do fechar parênteses. Se forem de tipos diferentes, os parênteses não estão correctamente colocados. Sugestão: utilize uma lista para representar uma expressão aritmética, na qual os parênteses são representados por cadeias de caracteres (contendo um único carácter).

21. Escreva a seguir a cada expressão o resultado produzido pelo avaliador de Scheme (suponha que as expressões são avaliadas sequencialmente). Desenhe o diagrama de caixas e ponteiros correspondente. Identifique o lixo que é recolhido pelo mecanismo de recolha de lixo ("garbage collection").

```
>(define a '(1 2))
> (define b '(3 (4 5)))
> (define c (cons a (cdr b)))
```

```
> c
> (set-car! b a)
> b
> (equal? b c)
> (eq? b c)
```

9.2.1 Definição de tipos mutáveis

1. Uma fila de prioridades é uma estrutura mutável que é composta por um certo número de filas, cada uma das quais associada a uma determinada prioridade.

   Suponha que desejava criar uma fila de prioridades com duas prioridades, urgente e normal. Novos elementos são adicionados à fila indicando a sua prioridade e são colocados no fim da fila respectiva. Os elementos são removidos da fila através da remoção do elemento mais antigo da fila urgente. Se a fila urgente não tiver elementos a operação de remoção remove o elemento mais antigo da fila normal. Existe uma operação para aumentar a prioridade a qual remove o elemento mais antigo da fila normal e coloca-o como último elemento da fila urgente.

   (a) Especifique as operações básicas para o tipo fila de prioridades (com prioridades urgente e normal).

   (b) Escolha uma representação interna para o tipo fila de prioridades (com prioridades urgente e normal). Sugestão: use directamente as filas apresentadas nas aulas teóricas.

   (c) Escreva em Scheme o procedimento cria-fila-prioridades que cria um objecto que corresponde a uma fila de prioridades. Sugestão: use directamente as operações das filas apresentadas nas aulas teóricas (nova-fila, coloca, retira, inicio, fila?, fila-vazia?).

2. A estrutura de informação DEQE (do Inglês "Double Ended Queue") corresponde a uma fila à qual novos elementos podem ser adicionados a qualquer das extremidades e elementos podem ser removidos de qualquer das extremidades. Defina as operações básicas para uma DEQE.

Capítulo 10

Programação com objectos

10.1 Exercícios de revisão

1. Diga a que corresponde o conceito de objecto.

2. Em programação orientada a objectos existe o conceito de classe e de subclasse, bem como a noção de herança. Diga o que são estes conceitos e dê um exemplo de uma situação em que é útil a sua utilização.

10.2 Exercícios de programação

1. Crie o objecto `garrafa-água` que tem os métodos `beber`, `encher` e `verificar-nivel`. Só deve ser possível beber e encher se o nível da água não descer abaixo de 0 ou subir além do seu volume inicial.

2. O procedimento `cria-conta` apresentado na página **??** permite efectuar depósitos e levantamentos, mas podemos efectuar depósitos e levantamentos negativos, dando origem a erros, como se demonstra na seguinte interacção:

   ```
   > (define conta-01 (cria-conta 500))
   > ((conta-01 'd) 200)
   700
   > ((conta-01 'd) -1000)
   -300
   ```

   Escreva um procedimento que não só corrige este problema mas também possibilita a consulta ao saldo da conta.

3. Modifique o procedimento `envia-mensagem`, apresentado na página **??**, de modo a que este aceite um número arbitrário de valores associados a uma mensagem. Para esta modificação, necessita do procedimento primitivo `apply`, o qual recebe um procedimento e uma lista e aplica o procedimento usando os elementos da lista como argumentos. Por exemplo, `(apply + '(1 2 3))` é equivalente a `(+ 1 2 3)`. O procedimento `apply` realiza o passo 7 do mecanismo de avaliação apresentado na Secção **??**.

4. Um *contador* é um procedimento com estado interno que mantém um valor de contagem e que recebe como argumento um dos seguintes símbolos: `incrementa`, `decrementa`, `valor-actual`, `poe-a-zero`. O valor devolvido pelo contador corresponde ao valor de contagem depois de efectuada a operação pedida: incrementar ou decrementar de uma unidade, devolver o valor actual sem o modificar, e colocar o valor a zero, respectivamente. Se o argumento recebido não for nenhum destes símbolos, o contador assinala um erro. Escreva um procedimento que recebe o valor inicial de contagem de um contador e cria o contador correspondente. Por exemplo,

```
> (define c1 (cria-contador 5))
> (c1 'incrementa)
6
> (define c2 (cria-contador 10))
> (c2 'decrementa)
9
> (c1 'poe-a-zero)
0
> (c2 'valor-actual)
9
```

5. Defina um procedimento `cria-contador-limitado` que recebe dois números inteiros, correspondendo ao limite inferior e superior do contador, e devolve um objecto que aceita as mensagens `inc` e `dec` que permitem, respectivamente, incrementar e decrementar o valor do contador e devolvem o valor do contador no final. Se se tentar decrementar o valor do contador para baixo do limite inferior ou para cima do limite superior deve ser dado um erro e o valor do contador não é alterado. O contador quando é criado tem como valor o limite inferior. Exemplo:

```
> (define c1 (cria-contador-limitado 3 7))
```

```
> (c1 'inc)
4
> (c1 'inc)
5
> (c1 'dec)
4
> (c1 'dec)
3
> (c1 'dec)
Limite inferior atingido
```

6. Defina um procedimento em Scheme, chamado `cria-estacionamento`, que simula o funcionamento de um parque de estacionamento.

   O procedimento `cria-estacionamento` recebe um inteiro que determina a lotação do parque e devolve um procedimento que pode receber como argumentos:

   - `'entra-carro`, corresponde à entrada de um carro no parque de estacionamento. Se o parque estiver cheio, gera uma indicação de erro e não actualiza o valor dos carros estacionados.
   - `'sai-carro`, corresponde à saida de um carro no parque.
   - `'num-lugares`, indica o número de lugares livres.

   Por exemplo,

```
> (define est-1 (cria-estacionamento 15))
> (est-1 'entra-carro)
> (est-1 'entra-carro)
> (est-1 'num-lugares)
13
> (est-1 'sai-carro)
> (est-1 'num-lugares)
14
```

7. Escreva um procedimento chamado `faz-lanca-notas` que devolve um procedimento com estado interno que permite simular o lançamento de notas para um aluno a uma determinada cadeira. Este procedimento pode receber como argumentos:

- um inteiro – atribui nota a cadeira (considere que este valor é sempre introduzido correctamente entre 0 e 20); no limite só podem ser lançadas duas notas positivas.
- `'nota` – devolve a nota lançada (a melhor nota, no caso de ter feito melhoria).
- `'passou-cadeira?` – devolve um valor lógico, consoante o aluno tenha passado ou não à cadeira.
- `'fez-melhoria?` – devolve um valor lógico, consoante o aluno tenha feito ou não melhoria de nota. Considera-se que um aluno fez melhoria de nota se já foram lançadas duas notas positivas.

Por exemplo,

```
> (define Aluno99999 (faz-lanca-notas))
> (Aluno99999 'passou-cadeira?)
#f
> (Aluno99999 14)
> (Aluno99999 'nota)
14
> (Aluno99999 17)
> (Aluno99999 'fez-melhoria?)
#t
> (Aluno99999 18)
"Lancamentos de nota esgotados"
```

8. (a) Desenhe o diagrama de ambientes criado pela definição do procedimento `cria-conta-ordenado` apresentado na página **??**.

   (b) Adicione ao seu diagrama os ambientes criados pela interacção:

```
> (define conta-ord-01 (cria-conta-ordenado 500 100))
> ((conta-ord-01 'levantamento) 300)
200
> ((conta-ord-01 'levantamento) 250)
-50
> ((conta-ord-01 'consulta-saldo))
-50
> ((conta-ord-01 'levantamento) 100)
Limite de levantamento excedido
```

(c) Com base no diagrama de ambientes que criou, explique a razão por que o método `levantamento` do procedimento da página **??** necessita de utilizar `((conta 'consulta-saldo))` para aceder ao saldo de `conta` e não pode ser substituído por

```
(levantamento
  (lambda (quantia)
     (if (>= (- saldo quantia) (- ordenado))
         ((conta 'levantamento) quantia)
         (error "Limite de levantamento excedido"))))
```

9. Defina uma classe que corresponde a uma urna de uma votação. A sua classe deve receber a lista dos possíveis candidatos e manter como estado interno o número de votos em cada candidato. Esta classe pode receber um voto num dos possíveis candidatos, aumentando o número de votos nesse candidato em um. Deve também permitir apresentar os resultados da votação.

10. Escreva o programa da Secção **??**, utilizando o objecto pilha definido na Secção **??**.

11. Defina o objecto fila, tendo em atenção o tipo de informação definido na Secção **??**.

12. Utilizando o tipo abstracto de informação ChamadaTelefónica, escreva um procedimento *cria-cartao* que simula o funcionamento de um cartão telefónico. As operações possíveis são:

    - Dado um determinado montante inicial, criar um cartão novo com esse montante
    - Dado um determinado montante, carregar um cartão existente com esse montante
    - Dada uma chamada, descontar o custo dessa chamada num determinado cartão.

    Por exemplo, supondo que o vector Tarifas tinha os seguintes elementos:

    ??????????

    mostra-se abaixo um exemplo de interacção:

```
> (define cartao-joao (cria-cartao 100))
```

```
> (define cartao-maria (cria-cartao 100))
> ((cartao-maria 'carrega) 100)
Saldo actual:  200
> ((cartao-maria 'desconta) '(local 5))
Saldo actual:  195
> ((cartao-joao 'desconta) '(regional 5))
Saldo actual:  90
```

Nota: Assumiu-se que a representação externa para chamada é uma lista, em que o primeiro elemento representa o tipo de chamada, e o segundo a duração.

13. Pretende-se definir a classe enquadramento para simular o comportamento dos ambientes em Scheme. A classe a definir tem as seguintes características:

Classe: `enquadramento`

Parâmetros: envolvente (opcional; quando presente representa o enquadramento envolvente)

Subclasse de: —

Variáveis de estado: `tab`, `envolvente`

Métodos:

- `valor(nome)`
  Devolve o valor associado a nome.
- `definir(nome, valor)`
  Faz o mesmo que (define nome valor).
- `atribuir!(nome, valor)`
  Faz o mesmo que (set! nome valor).

Com esta classe, a interacção

```
> (define E0 (cria-enquadramento))
> (define E1 (cria-enquadramento E0))
> (define E2 (cria-enquadramento E1))
> (envia-mensagem E0 'definir 'x 1)
> (envia-mensagem E1 'definir 'z 3)
> (envia-mensagem E2 'definir 't 5)
> (envia-mensagem E2 'definir 'x 2)
```

define a estrutura

a qual permite obter a seguinte interacção:

```
> (envia-mensagem E0 'valor 'x)
1
> (envia-mensagem E1 'valor 'x)
1
> (envia-mensagem E2 'valor 'x)
2
> (envia-mensagem E1 'valor 't)
valor: variavel não ligada
> (envia-mensagem e1 'atribuir! 'x 10)
> (envia-mensagem e0 'valor 'x)
10
> (envia-mensagem E0 'definir 'quadrado (lambda (x) (* x x)))
> ((envia-mensagem E0 'valor 'quadrado) 5)
25
```

Para definir a classe enquadramento utilizam-se os tipos ligação (para representar a associação de um valor com um nome) e tabela (para representar um conjunto de ligações), os quais têm as seguintes operações básicas:

Tipo ligação

faz-ligação: símbolo x universal -¿ ligação faz-ligação(nome, valor) devolve a ligação correspondente à associação de valor a nome.

nome-ligação: ligação -¿ símbolo nome-ligação(lig) devolve o nome da ligação lig.

valor-ligação: ligação -¿ universal valor-ligação(lig) devolve o nome da ligação lig.

Tipo tabela

nova-tabela: -¿ tabela nova-tabela() devolve a tabela vazia.

insere-ligação: ligação x tabela -¿ tabela insere-ligação (lig, tab) devolve a tabela resultante de inserir lig em tab. Se tab já contiver uma ligação com o mesmo nome que lig, esta ligação é substituída por lig.

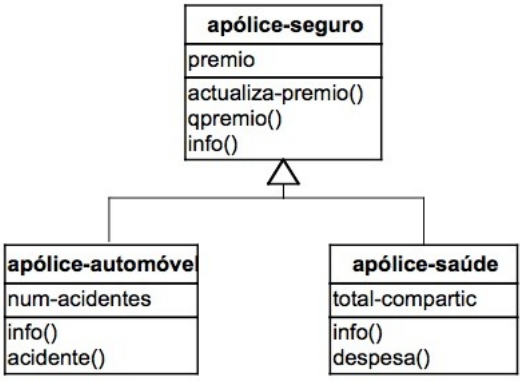
obtém-valor: tabela x nome -¿ universal obtém-valor(tab, nome), se existir em tab uma ligação cujo nome é nome, devolve o valor dessa ligação. Em caso contrário, devolve falso.

(a) Suponha que dispõe dos tipos ligação e tabela. Usando as operações básicas destes tipos, defina o procedimento cria-enquadramento.

(b) Escolha uma representação interna para o tipo ligação, e realize as operações básicas indicadas na alínea a).

(c) Escolha uma representação interna para o tipo tabela, e realize as operações básicas indicadas na alínea a).

14. Considere a seguinte hierarquia de classes que indica que há dois tipos de apólices de seguro - as apólices do ramo automóvel e as apólices de saúde.



Considere o seguinte exemplo de interacção:

```
> (define ap-001 (cria-apolice-automovel 1000))
> (mensagem ap-001 'info)
  premio:1000   num de acidentes:0
> (define ap-002 (cria-apolice-saude 200))
> (mensagem ap-002 'info)
  premio:200   total de comparticipações:0
> (mensagem ap-001 'acidente)
> (mensagem ap-001 'info)
  premio:1200.0   num de acidentes:1
> (mensagem ap-001 'acidente)
> (mensagem ap-001 'info)
  premio:1440.0   num de acidentes:2
> (mensagem ap-002 'acidente)
. apolice-seguro: operação inválida
> (mensagem ap-002 'despesa 50)
> (mensagem ap-002 'info)
  premio:200   total de comparticipações:24.0
```

(a) Considere a seguinte caracterização da classe apólice-seguro.

```
Nome: apólice-seguro
Parâmetros: prémio
Subclasse de: ---
Variáveis de estado: prémio
Métodos:
• actualiza-premio(novo-valor)
    Atribui o novo-valor indicado para o prémio.
• qpremio()
    Devolve o valor do prémio da apólice.
• info()
    Mostra o valor das variáveis de estado da classe.
```

Notando que a comunicação de acidente, para uma apólice de acidente, faz aumentar o prémio 20 por cento, preencha os espaços assinalados nas seguintes figuras:

Nome: apólice-automóvel

Parâmetros:

Subclasse de:

Variáveis de estado: Métodos: • info() Solicita a info da apólice-seguro e mostra o valor de num-acidentes.

Nome: apólice-saúde

Parâmetros:

Subclasse de:

Variáveis de estado: Métodos: • info() Solicita a info da apólice-seguro e mostra o valor de total-compartic • despesa(quantia) Se quantia ¡= 10, adiciona a total-despesas 60 por cento de (quantia – 10) (10 é o valor da franquia e 60 por cento a percentagem de comparticipação)

(b) Defina o procedimento mensagem que permite enviar mensagens aos objectos.

(c) Defina os procedimentos cria-apólice-seguro e cria-apólice-saúde.

15. Tendo em atenção o conceito de abstracção de dados, defina e implemente o tipo Conta Bancária. Uma conta bancária é caracterizada pelo seu saldo e pelas operações de depósito e de débito.

16. Suponha que o banco pretende cobrar a todas as contas o imposto anual num valor determinado anualmente (por exemplo Euro 30), criando o seguinte procedimento, que recebe uma lista de contas e o

imposto a cobrar, e que devolve o valor resultante da soma do novo saldo de todas as contas da lista.

```
(define (banco-cobra-imposto lst-contas imposto)
  (if (null? lst-contas)
      0
      (+ (conta-debito! (first lst-contas) imposto)
         (banco-cobra-imposto (rest lst-contas) imposto))))
```

O banco cria também o seguinte procedimento, que efectua o depósito dos juros em cada uma das contas. Este procedimento recebe como argumento uma lista de contas e a taxa de juros a usar; e devolve o valor resultante da soma do novo saldo de todas as contas da lista.

```
 (define (banco-deposita-juros lst-contas taxa)
  (if (null? lst-contas)
      0
      (+ (let* ((conta (first lst-contas))
                (saldo (conta-saldo conta)))
          (conta-deposito! conta (* saldo taxa)))
         (banco-deposita-juros (rest lst-contas) taxa))))
```

Repare que os dois procedimentos anteriores são semelhantes. Crie um procedimento de ordem superior capaz de implementar as duas operações.

17. Defina os procedimentos anteriores, banco-cobra-imposto e banco-deposita-juros, com base no procedimento desenvolvido na alínea anterior.