



1.1	1.2	2.1	2.2	2.3	2.4

**LEIC/Introdução à Programação I, 8 de Fevereiro de 2002****Duração máxima: 2 horas e 30 minutos; Com Consulta****Nome (legível):** \_\_\_\_\_ **Número** \_\_\_\_\_**Problema 1 (7.5 valores)**

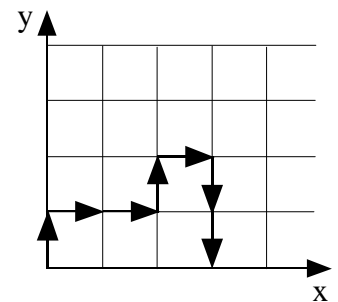
Um dispositivo robotizado desloca-se sobre uma grelha, obedecendo a uma lista de comandos do tipo: Norte, Este, Este, Norte, Este, Sul, Sul (ver figura). Não foi utilizado, mas também é reconhecido o comando Oeste.

Imaginando que a cada comando corresponde um passo unitário, então, no exemplo indicado, a distância entre os pontos de partida e chegada é de 3 unidades.

Num sistema de coordenadas  $xy$ , podemos utilizar a seguinte convenção:

*Comando Efeito*

- n a coordenada  $y$  aumenta 1 unidade;
- s a coordenada  $y$  diminui 1 unidade;
- e a coordenada  $x$  aumenta 1 unidade;
- o a coordenada  $x$  diminui 1 unidade.



**1.1** A distância  $d$  entre dois pontos, com as coordenadas  $x_1, y_1$  e  $x_2, y_2$ , determina-se a partir de:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Complete o procedimento **distancia-entre-pontos**, que calcula a distância entre dois pontos, cada um deles definido por uma lista com as suas coordenadas.

Exemplos:

```
> (distancia-entre-pontos (list 1 1) (list 1 6))
5
> (distancia-entre-pontos (list 1 1) (list 2 6))
5.0990195135927845
```

```
(define distancia-entre-pontos
  (lambda (p1 p2)
    (let ((p1-x (car p1))
          (p1-y (cadr p1))
          (p2-x (car p2))
          (p2-y (cadr p2)))
```

**1.2** Complete o procedimento **distancia-percurso**, que calcula a distância entre o ponto de partida, suposto na origem dos eixos e o ponto de chegada, do dispositivo robotizado. O percurso é especificado por uma lista de comandos.

Exemplos:

```
> (distancia-percurso '(n e e n e s s))
3
> (distancia-percurso '(n e e n e s s o))
2
> (distancia-percurso '(n e e n e s s X))
percurso-ambiguo
>
```

Sabe-se que este procedimento se baseia no procedimento **distancia-entre-pontos** e sabe-se ainda que o procedimento local **mais-um-passo** vai analisando, passo a passo, a lista que define o percurso e actualiza, de acordo com os comandos que encontra, as coordenadas  $x$  e  $y$ .

```
(define distancia-percurso
  (lambda (percurso)
    (letrec ((mais-um-passo
              (lambda (perc x y)
                (if (null? perc)
                    ; calcula distancia entre pontos

                    (cond ((equal? (car perc) 'n)
                          ; actualiza coordenadas x e y

                          ))
                    ))
      (mais-um-passo percurso 0 0))))
```

## Problema 2 Parte A (7.5 valores)

Num campeonato de futebol jogam várias equipas e pretende-se registar os resultados de todos os seus jogos, jornada a jornada. As equipas são identificadas apenas por um número, a começar pelo 1, continuando pelo 2, etc. Sendo  $n$  o número de equipas, o número de jornadas é igual a  $2 \times (n - 1)$ , pois, num campeonato, uma equipa joga com todas as outras equipas, uma vez em casa e outra fora. Em cada jornada, os resultados são registados da seguinte maneira:

```
equipa 1 1 2 (aqui, a equipa 1 perdeu o jogo por 1 a 2)
equipa 2 3 1 (a equipa 2 ganhou por 3 a 1)
equipa 3 2 1 ...
equipa 4 1 3
...
```

Suponha que uma jornada é representada num vector cujos elementos, um por cada equipa, são listas compostas pelo número da equipa e dois inteiros que representam, respectivamente, o número de golos marcados e sofridos pela equipa.

**2.1** Escreva em *Scheme* o selector **ver-jornada**, em que o único parâmetro é do tipo *jornada*.

```
; visualiza os resultados por jornada e por equipa, com o seguinte formato:
; equipa 1 - 1 2
; equipa 2 - 3 1
; equipa 3 - 2 1
; equipa 4 - 1 3
; ...
(define ver-jornada
  (lambda (jornada)
```

**2.2** Escreva em *Scheme* o construtor ***cria-jornada***, que devolve uma entidade do tipo *jornada*. Supor que inicialmente os golos marcados e sofridos atribuídos são: -1 -1.

```
(define cria-jornada
  (lambda (n-equipas)
```

## **Problema 2 Parte B** (5.0 valores)

**2.3** Considerando, por exemplo, um campeonato de 4 equipas, apresente graficamente a estrutura de dados utilizada para uma entidade do tipo *campeonato*. Justifique adequadamente a resposta.

**2.4** Escreva em *Scheme* o selector ***situacao-equipa-ate***.

```
; devolve a situação referente a uma equipa (um número de 1 até ...) desde a
; primeira jornada ate' ao final de uma jornada (n-jornada, um inteiro de 1 até ...)
; de um campeonato (do tipo campeonato).
;
; A situação devolvida será especificada por uma lista de 3 inteiros que definem,
; respectivamente:
;
; 1/ o número de pontos alcançados pela equipa até à data (vitória - 3 pontos;
; empate - 1 ponto; derrota - 0 pontos)
; 2/ o número de golos marcados
; 3/ o número de golos sofridos.
;
(define situacao-equipa-ate
  (lambda (campeonato equipa n-jornada)
```

**(Fim.)**

Só por curiosidade, seguem-se outros procedimentos que poderiam ser úteis na situação exposta:

```
; cria uma entidade do tipo campeonato, tendo em linha de conta a estrutura
; de dados escolhida para este efeito.
; certamente que este construtor utilizará o construtor cria-jornada,
; uma vez que um campeonato integra um certo número de jornadas, número que é
; função do número de equipas que jogam no campeonato.
(define cria-campeonato
  (lambda (n-equipas)
    ...

; introduz um resultado de um campeonato, referente a uma equipa, numa jornada.
; O resultado e' definido pelo numero de golos marcados e sofridos.
(define resultado!
  (lambda (campeonato equipa jornada marcados sofridos)
    ...

; visualiza os resultados de uma equipa, em todo o campeonato. Os resultados ainda
; não preenchidos são visualizados com o valor inicial: -1 -1
;
; tipo de visualização:
; equipa 2:
; jornada 1 - 2 3
; jornada 2 - 3 1
; ...
(define ver-equipa
  (lambda (campeonato equipa)
    ...
```