FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Licenciatura em Engenharia Informática e Computação

Introdução à Programação I

Exame de Recurso, 10 Fevereiro de 2001

DURAÇÃO MÁXIMA 2 horas e 30 minutos, com consulta

Aluno ______________________ Nº ______

Problema 1 (7.5 valores)

Observe com atenção a seguinte pirâmide de números inteiros.

```
                        1   ...........................   linha 1
                     1     1   ........................   linha 2
                  1     2     1   .....................   linha 3
               1     3     3     1   ..................   linha 4
            1     4     6     4     1   ...............   linha 5
         1     5    10    10     5     1   ............   linha 6
      1     6    15    20    15     6     1   .........   linha 7
   1     7    21    35    35    21     7     1   ......   linha 8
```

O primeiro e último número de cada linha é sempre 1. Os números intermédios são iguais à soma dos dois números mais próximos da linha anterior. Por exemplo, o número na 4ª posição da linha 8, o 35, é igual à soma dos números na 3ª e 4ª posições da linha 7, ou seja, 35 = 15 + 20.

A função *ponto-piramide* tem dois parâmetros, *lin* e *col*, e devolve o número que se encontra na linha *lin* e na coluna *col* da pirâmide. Por exemplo, *ponto-piramide (8, 4)* devolve 35.

1.1 Comece por apresentar uma definição recursiva (em notação matemática) para a função *ponto-piramide (lin, col)*, em que *lin* e *col* são garantidamente inteiros positivos. A função vale *0* quando *col* é maior que *lin*.

1.2 Escreva em Scheme o procedimento *ponto-piramide* correspondente à função apresentada na questão anterior, sabendo que os argumentos serão sempre inteiros positivos.

1.3 Escreva em Scheme o procedimento *linha-piramide* com o parâmetro *lin* que devolve uma lista com todos os pontos da linha *lin* da pirâmide. Por exemplo, *(linha-piramide 4)* devolve *(1 3 3 1)*.

Problema 2 **(7.5 valores)**

Um polinómio de grau n

$$p(x) = c_0 + c_1x + c_2x^2 + \ldots + c_nx^n \quad (n \geq 0) \quad (c_n \neq 0)$$

pode ser representado em Scheme pela lista de coeficientes

$$(c_0\ c_1\ c_2 \ldots c_n).$$

Por exemplo, o polinómio $p(x)=1+6x+2x^3$ seria representado pela lista

$$'(1\ 6\ 0\ 2).$$

2.1 Escreva em Scheme o procedimento *multiplica-polinómio*, que recebe como argumentos um polinómio *pol* (representado pela sua lista de coeficientes) e uma constante *x*, e devolve um novo polinómio com os coeficientes do primeiro polinómio multiplicados por *x*.

Exemplo de utilização:

```
➢ (define p1 '(1 6 0 2))
(1 6 0 2)
➢ (define p2 (multiplica-polinomio p1 2))
(2 12 0 4)
```

```
(define multiplica-polinomio
  (lambda (pol x)
```

2.2 Escreva em Scheme o procedimento *avalia-polinómio*, que recebe como argumentos um polinómio *pol* (representado pela sua lista de coeficientes) e um ponto *x*, e devolve o valor do polinómio *pol* no ponto *x*.

Sugestão: $c_0 + c_1x + ... + c_nx^n = c_0 + x\,(c_1 + ... + c_nx^{n-1})$.

Exemplo de utilização (no seguimento do exemplo anterior):

```
➢ (avalia-polinomio p1 1)
9
```

```
(define avalia-polinomio
  (lambda (pol x)
```

2.3 Escreva em Scheme o procedimento *cria-polinomio*, que recebe como argumentos os coeficientes de um polinómio, e devolve um procedimento com um argumento *x* que dá o valor do polinómio em *x*. Use o procedimento da alínea anterior.

Exemplo de utilização:

```
> (define p3 (cria-polinomio 1 6 0 2))
> (p3 1)
9
```

```
(define cria-polinomio
```

Problema 3 (5.0 valores)

Observe com atenção o diálogo entre o utilizador e o computador. Trata-se de uma sessão do jogo da vida. Na sessão, em primeiro lugar, é criado uma entidade *jogo da vida* com a designação *j*, em que 5 define a dimensão de um tabuleiro (neste caso, 5 linhas e 5 colunas) e 15 o número de caracteres + colocados aleatoriamente nesse tabuleiro. As restantes células são ocupadas com caracteres .

```
> (define j (faz-jogo-vida 5 15))
> (visu-jogo-vida j)
- + - + +
- + - + +
- - - - +
+ + + - +
+ + + + -
```

Quando o modificador *próximo!* é chamado, o valor de cada célula do tabuleiro é observado (só poderá ser + ou -), a fim de se calcular o seu próximo valor. Assim, se:

- Se o valor da célula for e se tiver 3 vizinhos +, o próximo valor será +.
- Se o valor da célula for + e se não tiver 3 vizinhos +, o próximo valor será .
- Nos casos restantes, o valor da célula mantém-se.

```
> (proximo! j)
actualizado
> (visu-jogo-vida j)
- - - + +
- - - - -
+ - - - +
+ - - - -
+ - - + -
```

```
> (proximo! j)
actualizado
> (visu-jogo-vida j)
- - - - -
- - - + +
- - - - -
- + - - -
- - - - -
```

```
> (proximo! j)
actualizado
> (visu-jogo-vida j)
- - - - -
- - - - -
- - - - -
- - - - -
- - - - -
```

```
> ...
```

Imagine que é dada a abstracção *jogo da vida*, com:

- *faz-jogo-vida*, com os parâmetros *dim* e *n*, em que *dim* define a dimensão do tabuleiro e *n* representa o número de células a preencher aleatoriamente com o caracter +.
- *sel-jogo-vida*, com os parâmetros *jogo*, *i* e *j*, que devolve o valor associado à célula do tabuleiro *jogo*, situada na linha *i* e na coluna *j*.
- *dim-jogo-vida*, com o parâmetro *jogo*, que devolve a dimensão do tabuleiro de *jogo* (o número de linhas ou colunas).
- *mod-jogo-vida!*, com os parâmetros *jogo*, *i*, *j* e *simb*, que altera o valor da célula do tabuleiro *jogo*, situada na linha *i* e na coluna *j*, para o valor *simb*.

3.1 Diga que estrutura de dados utilizaria para a abstracção do jogo da vida. Justifique a opção.

3.2 Escreva em Scheme o modificador *proximo!*, bem como os procedimentos auxiliares necessários. Não se esqueça que a abstracção *jogo da vida*, com os procedimentos associados, acima indicada, deve ser considerada dada, não se exigindo a sua escrita.

(Fim.)