Ficha 2

Programação Imperativa

Algoritmos Numéricos sobre inteiros

1. Podemos calcular o produto de um número m por um inteiro n através de um somatório de n parcelas constantes (e iguais a m).

   Assim
   $$n \times m = \sum_{i=1}^{n} m = \underbrace{m + m + \cdots + m}_{\text{n vezes}}$$

   Este cálculo pode ser efectuado somando `n` vezes a quantidade `m` a uma variável inicialmente com o valor `0`.

   Apresente uma definição da função `float multInt (int n, float m)` baseada nesta observação: a uma variável `r` (inicialmente com o valor `0`) será somado o valor de `m`, `n` vezes. Essa variável `r` vai ter os valores `0`, `m`, `2*m`, `3*m`, ..., e no final terá o valor desejado `n*m`.

2. Uma forma alternativa (e muito mais eficiente) consiste em aproveitar a representação binária dos inteiros (onde a multiplicação e divisão por 2 são pelo menos tão eficientes como a adição).

   Dados dois números `m` e `n` podemos construir uma tabela em que vamos dividindo (divisão inteira) `n` por `2` e multiplicando `m` por `2`. A primeira linha da tabela tem o valor original de `n` enquanto que a última corresponde a `n` ser `1`.
   Para obter o valor do produto de n por m basta somar os valores de `m` correspondentes às linhas em que `n` é impar.
   A tabela ao lado corresponde a um exemplo em que `n=81` e `m=423`.

	n	**m**
1	81	423
2	40	846
3	20	1692
4	10	3384
5	5	6768
6	2	13536
7	1	27072

   Se somarmos os valores de `m` para os quais `n` e impar (i.e., as linhas 1, 5 e 7) obtemos $423 + 6768 + 27072 = 34263 = 81 * 423$.

   Apresente uma definição alternativa da função `float multInt (int n, float m)` usando este processo.

3. O cálculo do máximo divisor comum entre dois números inteiros não negativos pode ser feito, de uma forma muito pouco eficiente, procurando de entre os divisores do menor deles, o maior que é também divisor do outro.

   Defina uma função `int mdc (int a, int b)` que usa esse método para calcular o máximo dicvisor comum entre dois números).

4. Uma forma alternativa de calcular o máximo divisor comum (mdc) baseia-se na seguinte propriedade demonstrada por Euclides: para a e b inteiros positivos,

$$mdc\ (a, b) = mdc\ (a + b, b)$$

Desta propriedade podemos concluir que:

$$\text{mdc } (a, b) = \begin{cases} mdc\ (a - b, b) & \text{Se } a > b \\ mdc\ (a, b - a) & \text{Se } a < b \\ a & \text{Se } a = b \end{cases}$$

O cálculo do máximo dicvisor comum pode ser feito por um processo repetitivo em que substituimos o maior dos argumentos pela diferença entre eles, até que um deles seja 0. Nessa altura, o valor do outro corresponde ao valor pretendido. Por exemplo, para calcularmos o máximo divisor comum entre 126 e 45 passaríamos pelos estados que se apresentam à direita.

a		**b**	
126		45	
81	(=126-45)	45	
36	(=81-45)	45	
36		9	(=45-36)
27	(=36-9)	9	
18	(=27-9)	9	
9	(=18-9)	9	
0	(=9-9)	9	

Apresente uma definição alternativa da função `int mdc (int a, int b)` que usa esse método.

5. Uma forma de melhorar o comportamento do algoritmo de Euclides consiste em substituir as operações de subtracção por operações de `%` (resto da divisão inteira). Repita o exercício da alínea anterior para essa variante do algoritmo.

6. A sequência de Fibonacci define-se como

$$fib\ (n) = \begin{cases} 1 & \text{Se } n < 2 \\ fib\ (n-1) + fib\ (n-2) & \text{Se } n \geq 2 \end{cases}$$

   (a) Apresente uma definição recursiva de uma função que `int fib (int n)` calcula o n-ésimo número desta sequência.

   (b) O cálculo do n-ésimo número de Fibonacci pode ser definido de uma forma mais eficiente (e iterativa) se repararmos que ele apenas necessita de conhecer os valores dos 2 valores anteriores. Apresente uma definição alternativa (e iterativa) da função da alínea anterior que calcula o n-ésimo número de Fibonacci, usando duas variáveis auxiliares que guardam os dois valores anteriores.