Ficha 2

Programação Imperativa

1 Memória e endereçamento

1. Diga justificando qual o resultado de executar o seguinte programa:

   ```
   int main () {
     int i, j, *a, *b;

     i=3; j=5;
     a = &i; b = &j;
     i++;
     j = i + *b;
     b = a;
     j = j + *b;
     printf ("%d\n", j);

     return 0;
   }
   ```

2. Considere a seguinte definição de uma função `void init (int a)`

   ```
   void init (int a) {
    a = 0;
   }
   ```

   Diga justificando qual o resultado de executar o seguinte código:

   ```
   int x;
   x = 3;
   init (x);
   printf("%d\n", x);
   ```

   Como modificaria a função (e a sua invocação) para que o resultado fosse `0`.

3. Defina uma função `void swap (.....)` que troca o valor de duas variáveis. Por exemplo, o código

   ```
   int x = 3, y = 5;
   swap(...);
   printf ("%d␣%d\n", x, y);
   ```

   deverá imprimir no ecran `5 3`.

2 Algoritmos Numéricos sobre inteiros

1. Uma forma de definir a multiplicação por um inteiro n é através de um somatório de n parcelas constantes.

   Assim
   $$n \times m = \sum_{i=1}^{n} m$$
   Esta definição corresponde à definição recursiva que se apresenta à direita.
   Apresente ums definição iterativa desta função.

   ```
   float mult (int n, float m) {
    float r;
    if (n>0)
      r = m + mult (n-1, m);
    else r = 0;

    return r;
   }
   ```

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).

   Se analisarmos a definição anterior em dois casos (caso em que o multiplicador é par ou ímpar), obtemos a seguinte definição:

   $$n \times m = \begin{cases} 0 & \text{Se } n = 0 \\ n/2 \times (m * 2) & \text{Se } n \text{ é par} \\ m + ((n-1)/2 \times (m * 2)) & \text{Se } n \text{ é ímpar} \end{cases}$$

   Que corresponde à definição recursiva que se apresenta abaixo.

   Apresente uma definição iterativa desta função.

   ```
   int mult (int n, float m) {
     float r = 0;
     if (n>0) {
       if (n % 2 != 0)
         r = r + m;
       r = r + mult (n/2, m+m);
     }
     return r;
   }
   ```

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.
   Quantas iterações faz o ciclo desta função para valores dos argumentos de `(1,1000)` e `(999,1000)`?

```
int mdc (int a, int b) {
  int d;

  if (a>b) d = b;
  else     d = a;

  while ((a % d != 0) ||
         (b % d != 0))
    d--;

  return d;
}
```

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:

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

   Que corresponde à definição recursiva que se apresenta à direita.
   Apresente ums definição iterativa desta função.

```
int mdc (int a, int b) {
  int r;
  if (a == b) r = a;
  else if (a > b)
    r = mdc (a-b, b);
  else r = mdc (a, b-a);
  return r;
}
```

5. Quantas iterações faz o ciclo da função que apresentou na alínea anterior para valores dos argumentos de `(1,1000)` e `(999,1000)`?

6. 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.

7. 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 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.