Ficha de Trabalho N.º 8

Objectivos: Recursividade

Soluções Propostas

1 - Calcule o valor do somatório apresentado a seguir, usando uma função recursiva (o valor de *n* deve ser pedido ao utilizador).

$$\sum_{i=1}^{n} i$$

```
int somat(int i)
{
    if (i==1)
       return 1;
    else
       return i+somat(i-1);
}
```

```
int main(void)
{
    int a1;
    a1=leitura_inteiro();
    printf("\n O Somatório dos inteiros entre 1 e %d: %d\n",a1,somat(a1));
    getchar();
}
```

Nota: a função de leitura pode ser a seguinte:

```
int leitura_inteiro()      // Função para leitura de um valor inteiro
{
    int n;
    printf("Qual o valor (inteiro)? ");
    scanf("%d",&n);
    return n;
}
```

Outra versão:

```
int somat_1_n(int i,int n)
{
    if (i==n)
      return n;
    else
      return i+somat_1_n(i+1,n);
}
```

```
int main(void)
{
     int a1;
     a1=leitura_inteiro();
     printf("\n Somatorio dos inteiros entre 1 e %d: %d\n",a1,somat_1_n(1,a1));
}
```

2 - Escreva uma função recursiva que permita calcular o factorial de um valor inteiro *n* (pedido ao utilizador). Faça um programa para testar a sua função.

```
#include <stdio.h>

int leitura_inteiro();      // Função para leitura de um valor inteiro
...
;
int factorial_v(int i) // Cálculo do factorial: passagem por valor
{
      if (i==1)
            return 1;
      else
            return i*factorial_v(i-1);
}

void factorial_r(int i, int *f) // Cálculo do factorial: passagem por referência
{
      if (i==1)
            *f = 1;
      else
      {
            factorial_r(i-1,f);
            *f= i* (*f);
      }
}

int main(void)
{
      int a1,fact;
      a1=leitura();
      factorial_r(a1, &fact);
      printf("\n (referencia) > factorial de %d: %d\n",a1,fact);
      printf("\n (valor) > factorial de %d: %d\n",a1,factorial_v(a1));
      getchar();
}
```

3 - Escreva um programa em C que inclua uma função recursiva chamada potencia que permita calcular x^n (*x* real, *n* inteiro positivo).

```
float potencia(float x, int i)
{
  if (i==1)
    return x;
  else
    return x * potencia(x,i-1);
}
```

```
int main(void)
{
 int a1;
 float r;
 a1=leitura_inteiro();
 r=leitura_real();
 printf("\n %.1f elevado a %d: %.1f\n",r,a1,potencia(r,a1));
}
```

4 - Calcule o valor do somatório apresentado a seguir, usando uma função recursiva.

$$\sum_{i=1}^{n} \frac{r^i}{i!}$$

Os valores de r (real) e n (inteiro) devem ser pedidos ao utilizador.

...

```
float leitura_real()    // Função para leitura de um valor real
{
 float x;
 printf("Qual o valor (real)? ");
 scanf("%f",&x);
 return x;
}
```

```
int leitura_inteiro()    // Função para leitura de um valor inteiro
{
 int x;
 printf("Qual o valor (inteiro)? ");
 scanf("%d",&x);
 return x;
}
```

```
float somatorio(int i,int n,float num,int den, float r)
{
  if (i==n)
    return num/den;
  else
    return num/den+ somatorio(i+1,n,r*num,den*(i+1),r);
}
```

```
int main(void)
{
      int vn; float vr;
      vn=leitura_inteiro();
      vr=leitura_real();
      printf("\n Somatório:  %.1f\n",somatorio(1,vn,vr,1,vr));
}
```

NOTA: Segue-se outra versão da função somatório. Adeque a respectiva chamada no programa.

```
float somatorio_v2(int i,int n,float t,float r)
{
      if (i==n)
            return t;
      else
            return t + somatorio_v2(i+1,n,t*r/(i+1),r);
}

int main(void)
{
      int vn; float vr;
      vn=leitura_inteiro();
      vr=leitura_real();
      printf("\n Somatório:  %.1f\n", somatorio_v2(1,vn,vr,vr));
}
```

5 - Elabore funções recursivas que permitam calcular o valor de cada um dos somatórios das alíneas a seguir. Faça a simulação da função elaborada.

a) $x = \sum_{i=1}^{n} [(i)!]^i$

```
float somat_a(int i,int n,int f)
{
      int j, p=1;
      for (j=1; j<=i; j++)
            p=p*f;
      if (i==n)
            return p;
      else
            return p + somat_a(i+1,n,f*(i+1));
}
```

Chama-se usando `x = somat_a(1,vn,1);`

b) $x = \sum_{i=1}^{n} i * (i+1)!$

```
float somat_b(int i,int n,int f)
{
      f = f * (i+1); // f<- (i+1)!
      if (i==n)
            return i * f; // último termo
      else
            return i * f +
      somat_b(i+1,n,f);
}
```

Chama-se usando `x = somat_b(1,vn,1);`

c) $x = \sum_{1=1}^{n} \frac{(i!)}{i}$

```
float somat_c(int i,int n,int f)
{
      if (i==n)
            return f;
      else
            return f + somat_c(i+1,n,f*i);
}

float somat_c_v2(int i,int n,int f) // versão 2
{
      f=f*i;
      if (i==n)
            return f/i;
      else
            return f + somat_c(i+1,n,f/i);
}
```

Chama-se usando `x = somat_c(1,vn,1);`

d) $x = \sum_{i=2}^{n} \frac{(i!)}{(i-1)!}$

```
float somat_d(int i,int n)
{
      if (i==n)
            return i;
      else
            return i + somat_d(i+1,n);
}
```

Chama-se usando `x = somat_d(2,vn);`

e) $x = \sum_{i=1}^{n} \frac{(i!)^2}{i+1}$

```
float somat_e(int i,int n,int f)
{
  if (i==n)
    return (float)(f*f)/(i+1);
  else
    return (float)(f*f)/(i+1) + somat_e(i+1,n,f*(i+1));
}
```

Chama-se usando `x = somat_e(1,vn,1);`

6 - Elabore várias funções para calcular o Fibonacci de um número:

a) uma, recursiva simples, aplicando directamente a própria definição de número de Fibonacci;

b) outra, iterativa;

c) ainda outra, recursiva pro, que usa um vector para armazenar os números de Fibonacci calculados anteriormente, que depois vão ser utilizados para o cálculo do seguinte.

Para cada uma das funções criadas, calcule e mostre o tempo de execução para o cálculo do Fibonacci de um número elevado (e.g. 35 ou 40), permitindo efecuar uma avaliação comparativa do tempo de execução e, consequentemente, do desempenho do algoritmo respectivo.

Possivelmente, vários factores externos podem alterar o tempo de execução (outros processos que estejam a correr no processador), pelo que, um melhoramento que se pode implementar é fazer várias medições para o tempo de execução de cada função e tomar valores médios, mostrando também o desvio máximo.

Obs. Para calcular o tempo de execução de cada tipo de implementação pode usar-se:

```
#include <time.h>
#include <sys/time.h>
```

Depois, colocar a definição dos dados:

```
struct timeval t1, t2;
double elapsedTime;
```

Quando se pretender iniciar a contagem, colocar o código:

```
// iniciar o cronómetro
gettimeofday(&t1, NULL);
```

Colocar depois o código de que se pretende calcular o tempo de execução, neste caso, a chamada à função:

```
//
// Código....
//
```

Quando se pretender finalizar a contagem e mostrar o tempo decorrido:

```
// Calcular e imprimir o tempo decorrido em milissegundos
elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;
elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;

printf("Texto informativo do processo que foi executado Terminado em %.9f milissegundos. \n\n",
elapsedTime);
```

```
// 6 - Elabore várias funções para calcular o Fibonacci de um número.
// a) uma, recursiva simples, aplicando directamente a própria definição de número de Fibonacci;
// b) outra, iterativa;
// c) ainda outra, recursivaPro, que usa um vector para armazenar os números de Fibonacci
//    anteriores que, depois, vão ser utilizados para o cálculo do seguinte.

Proposta de Resolução:

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>                // for gettimeofday()

// Definição recursiva do método Fibonacci
// o primeiro elemento é o elemento 1 que tem o valor 0
int fibonacciRec(int n)
{
    if (n == 0 || n == 1)
        return n;
    else
        return fibonacci( n - 1 ) + fibonacci( n - 2 );
}

// Definição não recursiva (iterativa) do método Fibonacci
// o primeiro elemento é o elemento 1 que tem o valor 0
int fibonacciIter(int n)
{
    int a, b, c;
    if (n == 0 || n == 1)
        return n; // caso base
    else
    {
        a = 0; // n-2
        b = 1; // n-1
        c = 0; // n
        for (int i = 2; i <= n; i++) // gerar os termos n>=3
        {
            c = b+a; // Fibonacci(n)=Fibonacci(n-1)+Fibonacci(n-2)
            a = b; // desloca n-2 para n-1
            b = c; // o n-1 passa a n
        }
        return c; // retorna o termo ordem n
    }
}

// Fibonacci com vector (versão FM)
//-------------------------------------------------
int fibonacciRecPro(int i, int vector[])
{
    if (i == 0) vector[0] = 0;
    if (i == 1) vector[1] = 1;

    //F(i) = F(i-1) + F(i-2); i >= 2
    if (i > 1)
    {
        if (vector[i - 2] == -1) vector[i - 2] = fibonacciRecPro (i - 2, vector);
        if (vector[i - 1] == -1) vector[i - 1] = fibonacciRecPro (i - 1, vector);
        vector[i] = vector[i - 2] + vector[i - 1];
    }
    return vector[i];
}

int fibonacciRec(int n);
int fibonacciIter(int n);
int fibonacciRecPro (int i, int vector[]);

int main(void)
{
    struct timeval t1, t2;
    double elapsedTime;

    int vector[100];
```

```
    int i, vn;
    for (i=0; i<100; i++)
        vector[i]=-1;

    printf("**** Calculo do Fibonacci de um numero ***\n");
    printf("** Utilizando uma função recursiva simples,
                    recursiva com vector e iterativa **\n\n");


    printf("Introduza o valor de n (inteiro): ");
    scanf("%d", &vn);

    // versão recursiva simples, utilizando a definição do Fibonacci de um número
    //
    // start timer
    gettimeofday(&t1, NULL);

    printf("\n\nVersão Recursiva Simples - O Fibonacci de %d = %d\n", vn, fibonacciRec(vn));

    // stop timer
    gettimeofday(&t2, NULL);

    // calcular e imprimir o tempo decorrido em milissegundos
    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
    elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms

    printf("vRecSimples: Executada em %.9f milissegundos. \n\n", elapsedTime);
    fflush(stdin);
    printf("\nDigite <Enter> para continuar...\n");
    getchar();
    getchar();


    // Versão recursiva com vector para guardar o Fibonacci dos n sucessivos números
    // start timer
    gettimeofday(&t1, NULL);

    printf("\n\nVersao Recursiva c/ vector PRO - O Fibonacci de %d = %d\n", vn,
                                        fibonacciRecPro(vn, vector));

    // stop timer
    gettimeofday(&t2, NULL);

    // calcular e imprimir o tempo decorrido em milissegundos
    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
    elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms

    printf("vRecursiva Pro: Executada em %.9f milissegundos. \n", elapsedTime);

    printf("\nDigite <Enter> para continuar...");
    getchar();

    // versao iterativa
    //
    // start timer
    gettimeofday(&t1, NULL);

    printf("Versão Iterativa - O Fibonacci de %d = %d\n", vn, fibonacciIter (vn));

    // parar o timer
    gettimeofday(&t2, NULL);

    // calcular e imprimir o tempo decorrido em milissegundos
    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
    elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms

    printf("Versão Iterativa: Executada em %.9f milisegundos. \n", elapsedTime);

    printf("\nDigite <Enter> para continuar...");
    getchar();

    return 0;
}
```