7ª Aula - Funções. Variáveis Dimensionadas (Vectores e Matrizes).

Programação

Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Função Factorial (I)

- Um outro exemplo muito frequente de função é a **função factorial**. Como se sabe esta pode ser escrita na forma:

$$n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1$$

```
int
factorial (int n1)
{
  int n2 ;
  n2 = 1;
  while (n1 > 1)
    {
      n2 = n2 * n1;
      --n1;
    }
  return n2;
}
```

(Ver Prog07_01.c)

- Comecemos pelo esqueleto da função;
- Indicar o **tipo** da função;
- Escrever o código da função...
- Declarar a variável usada e iniciá-la;
- Fazer um ciclo em que se escreve o resultado em **n2** e **n1** vai diminuindo de uma unidade até **1**;
- Finalmente indicar o valor de **retorno da função**;

Função Factorial (II)

- A partir da **definição de factorial** é fácil ver que

$$n! = n \times (n-1)! \quad \Longleftrightarrow \quad fac\,(n) = n \times fac\,(n-1)$$

  e que esta cadeia irá terminar quando **n = 0**, (ou **n = 1**), pois, ambos têm por resultado '**1**'.
- Isto é, podemos **definir recursivamente** a **função factorial** de um modo extremamente simples:
  - Se **n == 0** $\Rightarrow$ $n! = 1$;
  - Se **n > 0** $\Rightarrow$ $n! = n \times (n-1)!$
- Assim, temos para a **função factorial** na sua forma **recursiva**:

```
int
factorial (int n)
{
  if (n == 0) return 1;
  return n * factorial (n - 1);
}
```

(Ver Prog07_02.c)

- Seja a estrutura da função;
- Se **n == 0**, '**0!**' = '**1**' e sai;
- Se **n > 0**, '**n!**' = **n** $\times$ **(n - 1)!**'

Função Factorial (III)

- É fácil verificar que a **função factorial** a partir de números bastante pequenos **deixa de funcionar correctamente**:
- Como se sabe:
  - $12! = 479\,001\,600$
  - $13! = 6\,227\,020\,800$
- Como um **int** são **4 bytes** (incluindo o sinal)

$$4\,\text{bytes} = 4 \times 8\,\text{bits} = 32\,\text{bits} = 2^{32} = 4\,294\,967\,296$$

  podemos ver que $13! > 2^{32}$, logo, **não cabe**!
- Se fizermos, à mão, a conta:

$$6\,227\,020\,800 - 4\,294\,967\,296 = 1\,932\,053\,504$$

  e compararmos com resultado de **13!** obtido a partir do **Prog07_01/2.c** vemos que o **resultado** é aquela **diferença**!
- **Nota:** o que se obtém é o **resto da divisão** do número por 2^{32}. Alguns cuidados extra são necessário devidos aos valores positivos e negativos. Falaremos disso mais tarde.

Função Factorial (IV)

- Podemos ultrapassar, em parte, esta limitação de dois modos distintos sem utilizar modos especiais de computação:
  1. Usando **inteiros muito longos** (**long long int**) – ver **limits.h**:

     LLONG_MAX = 9223372036854775807
  2. Usando a **reais em dupla precisão** (**double**):

     $\mathbf{1.797693 \times 10^{308}}$
- Podem ver-se versões do programa **Prog07_02.c** adaptadas para '**long int**' (**Prog07_03.c**), para '**long long int**' (**Prog07_04.c**) e para '**double**' (**Prog07_05.c**).
- Note-se que presentemente '**long int**' pode comportar-se como '**int**' ou como como '**long long int**' dependendo dos sistemas.
- Como curiosidade, pode calcular-se a função factorial no sistema **maxima**[1] ou em qualquer outro sistema de computação algébrica (Computer Algebra System).

[1] Por exemplo calcular **factorial (1000);**

Variáveis Dimensionadas - Introdução

- Até aqui limitámo-nos a considerar variáveis às quais associávamos **um valor numérico**.

  Por exemplo: **i = 1;**, **x = 2.3;**, etc..
- Se nos limitássemos a estas variáveis, seria muito pouco prático guardar e manipular os dados de uma experiência em que tinhamos, por exemplo, **50 valores**.
- E se em vez de **50** tivessemos **1 milhão**?
- Obviamente, essa não é a solução adequada.
- Felizmente, existem mecanismos relativamente simples que nos permitem chegar a uma solução adequada.

Variáveis Dimensionadas - Introdução

Se quisermos **guardar** em casa, digamos, **20** números (representados por pedrinhas) em **caixas**, como fazemos?

1. Vamos ao supermercado, compramos caixas (azuis, por exemplo), colocamos **20 delas encostadas umas às outras** na estante e chamamos-lhes as '**caixas azuis**';
2. Depois, na **primeira** colocamos as **pedrinhas** correspondente ao **primeiro número** que queremos representar;
3. A seguir, passamos à **segunda** e fazemos o mesmo e assim sucessivamente até à última;
4. A partir daqui podemos referir-nos a esses valores como "**o primeiro**", "**o sétimo**", "**o décimo segundo**" das '**caixas azuis**'...

Em resumo, arranjámos uma **sequência de caixas**, as '**caixas azuis**', que colocámos em algum **sítio** (na estante), **numerámo-las** e passámos a **referirmo-nos** a elas pelos seus **números**.

Variáveis Dimensionadas ('Prog_06_05.c', '08_04.c')

A descrição feita no slide anterior não é mais do que a de uma **variável dimensionada** (**int ca[20];**) em que:

- A variável **ca** ('**caixas azuis**') é um **ponteiro** para a posição inicial da memória que lhe foi atribuída (**endereço**, '**morada**');
- O valor entre rectos **[20]** diz-nos que temos de reservar espaço para 20 '**int**'s (**20 caixas**), isto é, **80 bytes** (20×4);
- Assim, o **primeiro valor** guardado estará na posição de memória '**ca**', o segundo em '**ca + "o comprimento de 1 int"**', o terceiro em '**ca + 'o comprimento de 2 int'**' e assim sucessivamente;
- Isto é, o **primeiro** avança '**0**', o **segundo** '**1**', o **terceiro** '**2**', etc., então designaremos o **primeiro** valor por '**ca[0]**', o **segundo** por '**ca[1]**', o **terceiro** por '**ca[2]**', ... até ao último '**ca[19]**'.

Nota Muito Importante: A numeração das variáveis dimensionadas (de dimensão '**N**') começa **sempre** em '**0**' e vai até '**N - 1**'.

Variáveis Dimensionadas - Exemplo

```
x = x0;
for (i1 = 0 ; i1 <= i0 ; ++i1)
  x = r * x * (1.0 - x);
x_ref = x;
for (i1 = 0 ; i1 < imax ; ++i1)
  {
    x = r * x * (1.0 - x);
    vx[i1] = x;
    if (fabs (x - x_ref) < delta)
      break;
  }
  ++i1;
printf ("r=%.2lf , Qt: %ld - ",r,i1);
for (i2 = 0 ; i2 < i1 ; ++i2)
  printf (" %lf", vx[i2]);
printf ("\n");
```

Vamos alterar '**Prog05_08.c**', da função logística, para guardar, numa **variável dimensionada**, os **valores** das órbitas periódicas:

Prog05_10.c

- Deixamos a função estabilizar;
- Guardamos os valores da órbita no vector **vx[i1]**
- Incrementamos **i1** de uma unidade para ser igual ao periodo da órbita encontado;
- Escrevemos '**r**' e '**i1**';
- A seguir os valores da órbita;
- Passamos à linha seguinte.

Variáveis Dimensionadas - Notas Finais

- O programa **Prog05_11.c** é uma variante de **Prog05_10.c** em que a **escrita** no ecran passou a ser feita num **ficheiro**.
- Até aqui vimos como se trabalha com uma **variável dimensionada** com uma **única dimensão**.
- No entanto, podemos criar **variáveis dimensionadas** com muito mais **dimensões** e dos mais variados **tipos**. Exemplos:

  float bbb[4][23][2][3]; **int k[4][2];** **double x[4][234];**

  O modo como lidamos com estas variáveis é **análogo** ao que vimos para uma só dimensão.
- Para terminar, uma chamada de atenção **muito importante**: os **limites das variáveis não são testados**.
- Quando corremos um programa podemos, com uma certa facilidade, passar esses limites e escrever (ou ler) **noutras zonas da memória**. Os resultados são **imprevisíveis** e conduzem, muitas vezes, ao termo indevido do programa.