# 7ª Aula - Funções

# Programação
## Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@ist.utl.pt

Departamento de Física
Instituto Superior Técnico
Universidade Técnica de Lisboa

# Funções - Introdução

- Sempre que usámos, até aqui, a **função logística** tivemos de escrever **explicitamente** o seu cálculo:

  x = x * r * (1.0 - x);

- Quando isto é escrito uma única vez, não há necessidade de fazer mais, no entanto:
  - Se necessitarmos de a escrever diversas vezes ou
  - Se desejarmos isolar a função para a mais facilmente a alterar ou
  - Se desejarmos estruturar melhor o programa ou
  - Simplesmente para diminuir a probabilidade de erros de escrita

  pode ser mais simples (e conveniente) definir uma **função** que contenha aquele cálculo.

- Já vimos, como primeiro exemplo de uma **função**, a função '**main**' que retorna um **int**.

# Funções

- Uma **função** é uma entidade de **C** que tem um **nome**, um **tipo** (o seu retorno), **argumentos** (variáveis recebidas) e um **corpo** no qual deverá estar incluída instrução de retorno.
- A sua sintaxe é:

**tipo Nome_da_Função (Variáveis ...) {Corpo}**

- Seja então o seguinte exemplo (**Prog05_09.c** é uma versão adaptada de **Prog05_07.c** em que se introduz uma função e se substitui a **precisão simples**, **float**, pela **dupla**, **double**):

```
double
logistica (double x,
           double r)
{
  x = r * x * (1.0 - x);
  return x;
}
```

- Consideremos a função logística;
- O seu retornar vai ser do tipo **double**
- Vai receber como argumentos o valor de **x** e o parâmetro **r**;
- Faz o cálculo da iteração;
- **Retorna** o valor calculado.

# Função Factorial (I)

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

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

```
long int
factorial (long int n1)
{
  long 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**:

```
long int
factorial (long 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 pode ver:
  - $12! = 479\,001\,600$
  - $13! = 6\,227\,020\,800$
- Como um **long int** (ou 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 **resuldado** 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**):

     $\mathbf{1.797693 \times 10^{308}}$
- Versões do programa **Prog07_02.c** adaptadas a estas duas situações podem ser vistas nos programas **Prog07_03.c** e **Prog07_04.c**.
- 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);**