# 6ª Aula - Funções de Intervalo (III). Ciclo for.

# Leitura e Escrita em Ficheiros.

## Programação

## Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@ist.utl.pt

Departamento de Física
Instituto Superior Técnico
Universidade Técnica de Lisboa

# Ciclo com for (Prog05_07.c)

- Até aqui fizemos os ciclos usando a instrução **while**. Como vimos, a sintaxe do **while** é:

  **while (Condição) {'Corpo'}**

- Por exemplo, no caso da condição ser sobre um **int** (**int i1**) e só se efectuar enquanto **i1** $<=$ **20**, tem-se

  **i1 = 0; while (i1 <= 20) {... ++i1;}**

- O ciclo do **for** é análogo e tem a seguinte sintaxe:

  **for ( 'Início' ; Condição ; 'Incremento') {'Corpo'}**

- Assim, o exemplo anterior com a instrução **while** pode ser feito com a instrução **for** do seguinte modo:

  **for ( i1 = 0 ; i1 <= 20 ; ++i1) {'Corpo'}**

# Função Logística - Órbitas (I)

- Vimos atrás os **gráficos das órbitas** em função do parâmetro $r$.
- Podemos agora fazer um **programa** para calcular essas **órbitas** num certo intervalo $r \in [r_1, r_2]$, em que se incrementa o **parâmetro** $r$ de uma quantidade **dr**.
- Para tal temos de fazer um ciclo em que se efectuam, para cada $r$, as seguintes tarefas:
  1. Deixar a função **estabilizar**, isto é, calcular previamente um certo número de iterações;
  2. Guardar **o valor final** e imprimir os valores seguintes até ele se repetir.
- Para escrever este programa podemos usar ciclos **for** e usar **precisão dupla** (**double**), isto é, 8 bytes em vez dos 4 usados pela **precisão simples** (**float**).

# Função Logística - Órbitas (II) (Prog05_08.c)

```
for (r = r1 ; r <= r2 ; r += dr)
 {
  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);
    printf ("%f %f\n", r, x);
    if (fabs (x - x_ref) < delta)
     break;
   }
 }
```

- Calculam-se as **órbitas periódicas** com $r \in [r_1, r_2]$ incrementados de **dr**;
- Atribui-se a '**x**' o valor inicial '**x0**';
- Executam-se as '**i0**' iteradas;
- Toma-se para referência a última iterada '**x_ref**';
- Para testar a repetição, calculam-se novas iteradas até ao máximo **imax**;
- Imprimem-se o valor de '**r**' e de '**x**';
- Testa-se a diferença entre valores consecutivos com um erro de '**delta**' e, em caso **afirmativo**, pára-se o ciclo com a instrução **break;**

# Função Logística - Órbitas (III) (Prog05_08.c)

- Se quisermos **visualizar os pontos** obtidos a partir do programa **Prog05_08.c** num gráfico, em **unix**[1], podemos guardá-los num ficheiro, utilizando o **redireccionamento** do output:

  **./Prog05_08 > Prog05_08.txt**

- Depois pode usar-se o programa de gráficos '**gnuplot**' a visualizar o gráfico:

  **gnuplot**

  **gnuplot> plot "Prog05_08.txt" using** 1:2 **with** dots lt 3

  Experimentar substituir depois do '**with**' por: points lt 1 pt 9 lw 5

- Para mais indicações ver na página da cadeira:

  '**HOWTO − > gnuplot**'.

[1] Este procedimento pode igualmente ser feito em **Microsoft Windows** fazendo a execução do programa directamente na janelas de comandos.

# Como Guardar os Resultados de um Programa?

- Como se viu no programa anterior é escrita no ecran muita informação que, se não for **devidamente guardada**, se perde.
- Neste caso, podemos usar uma facilidade da **shell** de **Unix** que consiste em **redireccionar** o **output** do ecran para um ficheiro ('>'):

  **./Prog05_08 > 'ficheiro_onde_escrever.txt'**
- Note-se que também se pode **redireccionar** o **input** de um programa para uma file através do sinal inverso '<'.
- Do ponto de vista do **C**, '**escrever em**' significa abrir um **canal** para o qual é enviada a informação (o mesmo se passa na leitura).
- Quer a **escrita**, quer a **leitura** de **ficheiros**, do **ecran** ou de **outros locais** correspondem simplesmente a estabelecer um **canal** entre o **programa** que estamos a usar e a **entidade** para a qual queremos enviar (ou receber) os **dados**.

# Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) uma file é a função **fopen**:

  **FILE *fich** ;
  **fich** = **fopen** (" *nome_file* " , " *tipo_acesso* " );

- ***fich** é uma estrutura do tipo '**FILE**' que contém a informação referente ao ficheiro (e **fich** é o **ponteiro** para a **posição de memória** em que se encontra o **espaço reservado** a essa estrutura). No caso de **erro** retorna um ponteiro '**NULL**';
- *nome_file* é uma **string** com o **endereço** (absoluto ou relativo) do **ficheiro a aceder**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;
  - '**w**' abre para escrita (se a file já existe apaga o seu conteúdo);
  - '**t**' indica que o ficheiro é do tipo **texto**;
  - '**b**' indica que o ficheiro é do tipo **binário**;
  - etc...

# Abrir um Ficheiro – fopen

- Assim, por exemplo, para **abrir** para **escrita** um ficheiro de **tipo texto** '**dados.txt**' a instrução é:

  **fdados** = **fopen** ("**dados.txt**", "**wt**");

  em que **fdados** é o **descritor** do ficheiro (ou **stream**).
- A função que fecha uma canal aberto por é '**fopen**' é '**fclose**':

  **fclose** (**fdados**);
- Mas **não chega** abrir e fechar um ficheiro. É preciso **ler** e/ou **escrever** nesse ficheiro.
- Para **ler** e **escrever** em ficheiros de **tipo texto** (mais tarde falaremos dos **binários**) podemos usar **funções análogas** às usadas para interactuar com o terminar.
- Acrescenta-se um **f** (de **file**) ao nome dessas funções (**fprintf** e **fscanf**) e o seu **primeiro argumento** é o **ponteiro** retornado pela função **fopen**.

# Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
#include <stdio.h>
#include <math.h>
int main ()
{
 int i1 ;
 float x1 = 2722.0 ;
 FILE *fich1 ;
 fich1 = fopen ("data.txt", "wt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fprintf (fich1,"%d %f\n",i1,x1);
   x1 = sqrt (x1);
  }
 fclose (fich1);
 return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor da file;
- Fazer um ciclo (no qual se irá escrever na file);
- Declarar variáveis;
- Escrever na file os valores de 'i1' e de 'x1';
- Calcular valor seguinte de 'x1';
- Fechar o acesso à file.
- Incluir os "**headers**" '**stdio**' e '**math**';

# Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *fich1 ;
 fich1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fscanf (fich1,"%d %f\n",&i2,&x1);
   printf ("i2: %d ; x1: %f\n",i2,x1);
  }
 fclose (fich1);
 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;
- Abrir o ficheiro de texto '**data.txt**' para leitura;
- Declarar descritor da file;
- Declarar variáveis;
- Fazer um ciclo (no qual se irá ler da file);
- Ler da file os valores de 'i1' e de 'x1';
- Escrever no ecran os valores lidos;
- Fechar o acesso à file.

# Notas sobre Escrita e Leitura

- O programa '**Prog06_03.c**' junta os dois anteriores.
- Quando um programa é iniciado são abertos **três canais**, que estão orientados para o **terminal**, e cujos **descritores** são:
  - **stdin**: leitura;
  - **stdout**: escrita;
  - **stderr**: escrita de mensagens;
- Quando escrevemos no **ecran** estamos a usar implicitamente o canal **stdout**:

  **printf** ("Estou a escrever no terminal.\n");
- O **mesmo resultado** teríamos ao escrever:

  **fprintf** (**stdout**, "Estou a escrever no terminal.\n");
- Resultados idênticos se teriam para as leituras feitas a partir do terminal com **scanf** (...) ou **fscanf** (**stdin**, ...);
- No programa '**Prog06_04.c**' vemos como se pode utilizar o **retorno** do '**fscanf**' para **terminar** a leitura.