6ª Aula - Leitura e Escrita em Ficheiros (II). Variáveis Aleatórias. Funções

Programação

Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **fopen**:

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

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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**';

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;
  - '**w**' abre para escrita (se o ficheiro já existe apaga o seu conteúdo);

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;
  - '**w**' abre para escrita (se o ficheiro já existe apaga o seu conteúdo);
  - '**t**' indica que o ficheiro é do tipo **texto**;

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;
  - '**w**' abre para escrita (se o ficheiro já existe apaga o seu conteúdo);
  - '**t**' indica que o ficheiro é do tipo **texto**;
  - '**b**' indica que o ficheiro é do tipo **binário**;

Abrir um ficheiro – fopen

- A função de **C** para criar (ou abrir) um ficheiro é **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** ou a **criar**;
- *tipo_acesso* é uma **string** que define o **modo de acesso**:
  - '**r**' abre para leitura;
  - '**w**' abre para escrita (se o ficheiro 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:

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**");

.

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**).

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**);

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**.

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.

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** (**descritor**) retornado pela função **fopen**.

Exemplo de Escrita num Ficheiro (Prog06_01.c)

- Iniciar o programa;

```
int main ()
{




 return 0;
}
```

Exemplo de Escrita num Ficheiro (Prog06_01.c)

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;

```
int main ()
{

 fich1 = fopen ("data.txt", "wt");

 return 0;
}
```

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
int main ()
{


  FILE *fich1 ;
  fich1 = fopen ("data.txt", "wt");




  return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor do ficheiro;

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
int main ()
{


  FILE *fich1 ;
  fich1 = fopen ("data.txt", "wt");
  for (i1 = 0 ; i1 < 10 ; ++i1)
    {



    }

  return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
int main ()
{
 int i1 ;
 float x1 = 2722.0 ;
 FILE *fich1 ;
 fich1 = fopen ("data.txt", "wt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {



  }

 return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
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);

  }

 return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;
- Escrever no ficheiro os valores de 'i1' e de 'x1';

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
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);
  }

 return 0;
}
```

- Iniciar o programa;
- Abrir o ficheiro de texto '**data.txt**' para escrita;
- Declarar descritor do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;
- Escrever no ficheiro os valores de 'i1' e de 'x1';
- Calcular valor seguinte de 'x1';

Exemplo de Escrita num Ficheiro (Prog06_01.c)

```
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 do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;
- Escrever no ficheiro os valores de 'i1' e de 'x1';
- Calcular valor seguinte de 'x1';
- Fechar o acesso ao ficheiro.

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 do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;
- Escrever no ficheiro os valores de 'i1' e de 'x1';
- Calcular valor seguinte de 'x1';
- Fechar o acesso ao ficheiro.
- Incluir os "**headers**" '**stdio**' e '**math**';

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 do ficheiro;
- Fazer um ciclo (no qual se irá escrever no ficheiro);
- Declarar as variáveis;
- Escrever no ficheiro os valores de 'i1' e de 'x1';
- Calcular valor seguinte de 'x1';
- Fechar o acesso ao ficheiro.
- Incluir os "**headers**" '**stdio**' e '**math**';

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

- Iniciar o programa;

```
int main ()
{




 return 0;
}
```

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{



 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{


 f1 = fopen ("data.txt", "rt");




 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;
- Abrir o ficheiro de texto '**data.txt**' para leitura;

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{


 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");









 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;
- Abrir o ficheiro de texto '**data.txt**' para leitura;
- Declarar descritor do ficheiro;

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");




 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;
- Abrir o ficheiro de texto '**data.txt**' para leitura;
- Declarar descritor do ficheiro;
- Declarar as variáveis;

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {



  }

 return 0;
}
```

- Iniciar o programa;
- Incluir os "**header**" do **input/output**;
- Abrir o ficheiro de texto '**data.txt**' para leitura;
- Declarar descritor do ficheiro;
- Declarar as variáveis;
- Fazer um ciclo (no qual se irá ler do ficheiro);

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fscanf (f1,"%d %f\n",&i2,&x1);

  }

 return 0;
}
```

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

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fscanf (f1,"%d %f\n",&i2,&x1);
   printf ("i2:%d ; x1:%f\n",i2,x1);
  }

 return 0;
}
```

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

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fscanf (f1,"%d %f\n",&i2,&x1);
   printf ("i2:%d ; x1:%f\n",i2,x1);
  }
 fclose (f1);
 return 0;
}
```

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

Exemplo de Leitura dum Ficheiro (Prog06_02.c)

```
#include <stdio.h>
int main ()
{
 int i1, i2 ;
 float x1 ;
 FILE *f1 ;
 f1 = fopen ("data.txt", "rt");
 for (i1 = 0 ; i1 < 10 ; ++i1)
  {
   fscanf (f1,"%d %f\n",&i2,&x1);
   printf ("i2:%d ; x1:%f\n",i2,x1);
  }
 fclose (f1);
 return 0;
}
```

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

Ficheiro: Exemplos de Escrita e Leitura

- O programa '**Prog06_03.c**' junta os dois anteriores.

Ficheiro: Exemplos de Escrita e Leitura

- O programa '**Prog06_03.c**' junta os dois anteriores.
- No programa '**Prog06_04.c**' vemos como se pode utilizar o **retorno** do '**fscanf**' para **terminar** a leitura.

Ficheiro: Exemplos de Escrita e Leitura

- O programa '**Prog06_03.c**' junta os dois anteriores.
- No programa '**Prog06_04.c**' vemos como se pode utilizar o **retorno** do '**fscanf**' para **terminar** a leitura.
- Podemos agora alterar o programa da função logística '**Prog05_08.c**', que escrevia os resultados no ecran, para os guardar num ficheiro (ver '**Prog05_13.c**').

Notas sobre Escrita e Leitura

- Quando um programa é iniciado são abertos **três canais**, que estão orientados para o **terminal**, e cujos **descritores** são:

Notas sobre Escrita e Leitura

- 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;

Notas sobre Escrita e Leitura

- 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");
```

Notas sobre Escrita e Leitura

- 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");

Notas sobre Escrita e Leitura

- 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**, ...);

Variáveis Aleatórios - Introdução

- Em certos problemas é necessário utilizar **sequências aleatórias**, isto é, uma sucessão de números escolhidos **ao acaso**.

Variáveis Aleatórios - Introdução

- Em certos problemas é necessário utilizar **sequências aleatórias**, isto é, uma sucessão de números escolhidos **ao acaso**.
- Na verdade, quando se fala em "**escolhidos ao acaso**", referimo-nos a **processos deterministas** que geram sucessões **aparentemente** aleatórias.

Variáveis Aleatórios - Introdução

- Em certos problemas é necessário utilizar **sequências aleatórias**, isto é, uma sucessão de números escolhidos **ao acaso**.
- Na verdade, quando se fala em "**escolhidos ao acaso**", referimo-nos a **processos deterministas** que geram sucessões **aparentemente** aleatórias.
- Estas **sequências pseudo-aleatórias** são obtidas a partir de **funções de intervalo** em zona particulares dos **parâmetros**.

Variáveis Aleatórios - Introdução

- Em certos problemas é necessário utilizar **sequências aleatórias**, isto é, uma sucessão de números escolhidos **ao acaso**.
- Na verdade, quando se fala em "**escolhidos ao acaso**", referimo-nos a **processos deterministas** que geram sucessões **aparentemente** aleatórias.
- Estas **sequências pseudo-aleatórias** são obtidas a partir de **funções de intervalo** em zona particulares dos **parâmetros**.
- Uma vez que os computadores são **deterministas** é necessário, cada vez que se **inicia um programa**, dar um **ponto de partida diferente** à sequência aleatória (caso contrário, começariamos sempre no mesmo sítio, o que só é bom na fase de testes).

Variáveis Aleatórios - Introdução

- Em certos problemas é necessário utilizar **sequências aleatórias**, isto é, uma sucessão de números escolhidos **ao acaso**.
- Na verdade, quando se fala em "**escolhidos ao acaso**", referimo-nos a **processos deterministas** que geram sucessões **aparentemente** aleatórias.
- Estas **sequências pseudo-aleatórias** são obtidas a partir de **funções de intervalo** em zona particulares dos **parâmetros**.
- Uma vez que os computadores são **deterministas** é necessário, cada vez que se **inicia um programa**, dar um **ponto de partida diferente** à sequência aleatória (caso contrário, começariamos sempre no mesmo sítio, o que só é bom na fase de testes).
- Um modo simples de termos um **ponto sempre diferente** é usarmos o **instante em que o programa começa** para definir o **ponto de partida** da sequência.

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

  srand (**time** (**NULL**));

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

  srand (**time** (**NULL**));

- **int rand** (**void**);

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

  srand (**time** (**NULL**));

- **int rand** (**void**);

  Não tem argumentos e retorna um inteiro entre **0** e **RAND_MAX** (usualmente, 2147483647). Para um **double** em $[0, 1]$:

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

  srand (**time** (**NULL**));

- **int rand** (**void**);

  Não tem argumentos e retorna um inteiro entre **0** e **RAND_MAX** (usualmente, 2147483647). Para um **double** em $[0, 1]$:

  x = ((**double**) **rand** ()) / ((**double**) **RAND_MAX**);

Variáveis Aleatórios - Funções

Basicamente, duas funções em **C** são necessárias para obtermos uma **sequência aleatória** (essas funções estão definidas em '**stdlib.h**'):

- **void srand** (**unsigned int seed**);

  Serve para definir **internamente** em que ponto se **inicia** a sequência de números aleatórios (em geral, **usa-se uma só vez**);
  Para se ter **valores diferentes**, cada vez que se corre o programa, é usual dar-lhe como argumento o retorno da função '**time**' (o instante actual) que se encontra definida em '**time.h**':

  srand (**time** (**NULL**));

- **int rand** (**void**);

  Não tem argumentos e retorna um inteiro entre **0** e **RAND_MAX** (usualmente, 2147483647). Para um **double** em $[0, 1]$:

  x = ((**double**) **rand** ()) / ((**double**) **RAND_MAX**);

Ver '**Prog08_01/02/12.c**' para '**int**' e '**Prog08_03.c**' para '**double**'.

Funções - Introdução

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

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

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:

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

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 alterar mais facilmente ou

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 alterar mais facilmente ou
  - Se desejarmos estruturar melhor o programa ou

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 alterar mais facilmente ou
  - Se desejarmos estruturar melhor o programa ou
  - Simplesmente para diminuir a probabilidade de erros de escrita

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 alterar mais facilmente 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.

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 alterar mais facilmente 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 a instrução de retorno.

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 a instrução de retorno.
- A sua sintaxe é:

tipo Nome_da_Função (Variáveis ...) {Corpo}

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 a 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**):

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 a 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**):

```
logistica (
                    )
{


}
```

  - Consideremos a função logística;

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 a 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 (
                    )
{


}
```

- Consideremos a função logística;
- O seu retornar vai ser do tipo **double**

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 a 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)
{



}
```

- Consideremos a função logística;
- O seu retornar vai ser do tipo **double**
- Vai receber como argumentos o valor de **x** e o parâmetro **r**;

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 a 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);

}
```

- 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;

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 a 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.