Ficha de Trabalho N.º 7

Versão 2024/25

Objetivos: Uso de Ficheiros.

Conceitos Necessários à Resolução da Ficha

Ficheiros - Introdução

A utilização de ficheiros surge naturalmente da necessidade de, ao desligar o computador, manter dados contidos na memória *RAM* (memória volátil). Para que não sejam perdidos, recorre-se ao armazenamento em memória não volátil (disco), ficando assim guardados de forma permanente.

Os próprios resultados produzidos por um programa podem ser de grandes dimensões, podendo simplesmente não ser visualizáveis no monitor e muito menos analisados.

Por outro lado, na maioria das aplicações também não é praticável inserir manualmente os dados: estes encontram-se em ficheiros de onde são lidos para processamento.

Em *C* um ficheiro não é mais que uma sequência de *Bytes*.

Pode ser fonte de dados (*input*) para o programa ou destino de dados gerados pelo programa (*output*).

O conceito de STREAM
Stream* é um conjunto sequencial de caracteres: um conjunto de *Bytes

Ao falarmos em ficheiros, estamos a pensar em *streams* correspondentes a conjuntos de dados existentes em suporte magnético/*flash*.

A utilização de *streams* para entrada e/ou saída de dados é *Device Independent*.
Independentemente do tipo de periférico usado (ficheiro em disco, teclado, monitor, impressora, porta USB, etc), o C processa todas as entradas e saídas de dados através de *streams*.

OBSERVAÇÕES

- O nome de um ficheiro é armazenado numa *string* e deve representar fielmente o nome do ficheiro, tal como é visto pelo sistema operativo.
- O nome do ficheiro pode estar numa *string* constante ou num apontador para uma *string* algures na memória.

NOTA: Se o ficheiro for indicado dentro do programa, ter em atenção a existência do carácter especial '\':

EXEMPLO: o ficheiro C:\tmp\Dados.dat deverá ser escrito dentro de um programa como "C:\\tmp\\Dados.dat".Se a leitura for feita a partir do teclado, deve ser colocada apenas uma \

MODOS DE ABERTURA DE UM FICHEIRO

- "r" – *read* - abertura de um ficheiro para leitura. Caso não possa abrir (por não existir ou não ter permissões), a função **fopen** devolve NULL.
- "w" – *write* - abertura de um ficheiro para escrita. Se já existir um ficheiro com o mesmo nome, este é apagado e é criado um novo (vazio). Caso não possa criar (por o nome ser inválido, não haver espaço em disco, questões de protecção do dispositivo, etc.) devolve NULL.
- "a" – *append* - abertura de um ficheiro para acrescento aos dados já existentes. Se não existir, é criado um novo ficheiro.

Abertura de um ficheiro para leitura e escrita

Além dos 3 modos básicos de abertura referidos, existe ainda a possibilidade de abrir um ficheiro de forma a permitir, simultaneamente, operações de leitura e escrita, colocando um sinal + a seguir ao modo:

- "r+" – Se o ficheiro não existir, é criado. Se já existir, os novos dados são colocados a partir do início do ficheiro, apagando os dados anteriores.
- "w+" – Se o ficheiro não existir, é criado. Se já existir, é apagado e criado um novo ficheiro com o mesmo nome.
- "a+" – Se o ficheiro não existir, é criado. Se já existir, os novos dados são colocados a partir do final do ficheiro.

Abertura de ficheiros

A abertura de um ficheiro permite associar um ficheiro existente num suporte magnético ou de outro tipo, a uma variável do programa.

Para se poder utilizar um ficheiro, tem que se declarar uma variável do tipo FILE (mais propriamente, um apontador para o tipo FILE).

O tipo FILE encontra-se definido no ficheiro <stdio.h>.

A abertura de um ficheiro é realizada usando a função fopen() cujo protótipo se encontra no ficheiro <stdio.h>.

SINTAXE
FILE* fopen(const char* fileName, const char* mode)

- **filename** é uma ***string*** contendo o nome físico do ficheiro
- **mode** é uma ***string*** que contém o modo de abertura do ficheiro

Caso exista algum erro de abertura, a função devolve NULL

EXEMPLO

```
…
FILE *f = fopen(fileName, "r");
 if (f == NULL)
 {
  printf("\nProblemas na abertura do ficheiro! \n");
  return NULL;
 }
…
```

Se não houve problemas com a abertura do ficheiro cujo nome é dado na variável **fileName**, **f** fica a apontar para esse ficheiro (para leitura).
Se houve problemas, **f** fica a apontar para "nada".

EXERCÍCIO

O bloco de código acima faz parte de uma função.
a) Qual o tipo da variável fileName?
b) De que tipo pode ser o resultado fornecido pela função?

a) char* fileName;
b) Apontador para um ficheiro (é devolvido um endereço).

Modo de texto e modo binário

- Em caso de omissão do tipo de ficheiro, a abertura de um ficheiro é realizada considerando-o como ficheiro de texto.
- Para abrir um ficheiro em modo binário, é necessário acrescentar b ao modo de abertura (Exemplo: "rb", "wb", "ab", "a+b", etc.).

Um ficheiro é considerado de texto quando é constituído por caracteres por nós reconhecíveis. Normalmente, letras do alfabeto, dígitos, outros caracteres usados na escrita corrente ($, %, &, [, #, :, etc.) e ainda pelos separadores "espaço em branco", *Tab* e *New Line*: normalmente os caracteres existentes no nosso teclado. Regra geral um ficheiro de texto é formatado somente pelo caracter *New Line* que indica onde termina cada linha.

Os ficheiros binários podem ser constituídos por qualquer dos caracteres existentes na tabela ASCII, incluindo caracteres de controlo, caracteres especiais e mesmo caracteres sem representação visível (como o carácter terminador \0).

A maior diferença entre os dois processamentos está no tratamento do carácter "New Line" que em sistemas MS é constituído não por um, mas por dois caracteres: *Carriage Return* (13) e *Line Feed* (10).

Abertura de ficheiros – Síntese

Modo de abertura	Descrição	Permite leitura	Permite escrita	Ficheiro não existe	Ficheiro já existe	Posição inicial
r	Leitura	Sim	Não	NULL	OK	Início
w	Escrita	Não	Sim	Cria	Recria	Início
a	Acrescento	Não	Sim	Cria	OK	Fim
r+	Ler/escrever	Sim	Sim	Cria	Permite alterar dados	Início
w+	Ler/escrever	Sim	Sim	Cria	Recria	Início
a+	Ler/escrever	Sim	Sim	Cria	Permite acrescentar dados	Fim

Fecho de ficheiros

O fecho de um ficheiro retira a ligação entre a variável e o ficheiro.

Antes do fecho, são gravados todos os dados que ainda possam existir em *buffers* associados ao ficheiro.

É libertada a memória previamente alocada pela função fopen.

SINTAXE
```
int fclose(FILE* ficheiro);
```

- **A função devolve 0 em caso de sucesso.**
- **Em caso de erro, retorna a constante simbólica EOF (-1).**

Observação: Embora os ficheiros sejam automaticamente fechados quando uma aplicação termina, é uma boa prática fechá-los por programação, evitando desse modo problemas graves que podem surgir ao desligar subitamente o computador. Neste caso, os dados presentes em *buffers* não irão ser colocados no ficheiro e este pode não ficar estável, podendo chegar a ocorrer perda de sectores do disco.

Processamento de ficheiros de texto

Leitura e escrita de caracteres em ficheiro

A função mais utilizada para ler caracteres de um ficheiro é a função *fgetc* () - *file get char*

```
int fgetc (FILE* ficheiro)
```

A função *fgetc()* lê um carácter do ficheiro, passado por parâmetro. Se já não houver nenhum carácter no ficheiro, devolve a constante simbólica EOF (-1).

Todas as funções de leitura se posicionam automaticamente a seguir ao valor lido do ficheiro. O mesmo se aplica às funções de escrita.

A escrita de caracteres em ficheiro pode ser feita recorrendo à função *fputc() - file put char*

```
int fputc (int ch, FILE* ficheiro)
```

que escreve o carácter *ch* no ficheiro.

É possível ler ou escrever dados de modo formatado em ficheiros abertos em modo de texto, usando as funções *fscan()* e *fprint()*, colocando apenas mais um parâmetro inicial, correspondente ao ficheiro onde o processamento irá ser realizado.

Processamento de ficheiros binários

Quando falamos de ficheiros binários referimo-nos normalmente a ficheiros que não são de texto e correspondem, usualmente, a ficheiros de dados, ficheiros executáveis, etc.

Nos ficheiros binários não faz sentido ler uma linha. Esses ficheiros não estão organizados por linhas, como é habitual nos ficheiros de texto.

As funcionalidades exigidas à linguagem para tratamento de ficheiros binários destinam-se a realizar operações por Acesso Direto – só utilizadas em ficheiros abertos em modo binário.

O Acesso Direto é normalmente associado ao processamento de dados, os quais são escritos em blocos da memória para o disco e lidos em blocos do disco para a memória.

As funções de leitura e escrita que permitem Acesso Direto são, respectivamente,
- fread
- fwrite

Leitura e escrita de *strings* em ficheiro

A leitura de linhas a partir de um ficheiro de texto pode ser realizada através da função função *fgets* () - *File Get String*

```
char* fgets (char* str, int n, FILE* ficheiro)
```

Esta função coloca em str a *string* lida do ficheiro. Termina a leitura da *string* quando encontra um '\n' no ficheiro ou quando já tiver lido n-1 caracteres, pois reserva o outro para '\0'.

A função devolve a *string* lida apontada por str, ou NULL em caso de erro ou End of File.

Mantém nas *strings* os '\n' lidos do ficheiro.

A escrita de *strings* em ficheiro pode realiza-se através da função *fputs* () - *File Put String*

```
int fputs (const char* str, FILE* ficheiro)
```

A função *fputs*() devolve um valor não negativo caso a escrita seja realizada com sucesso, ou EOF caso contrário.

Escrita de blocos em ficheiros binários – função *fwrite*

SINTAXE
```
size_t fwrite (const void* ptr, size_t size, size_t nElements, FILE* file);
```

- `ptr` é um apontador para `void` (para qualquer tipo) e contém o endereço de memória daquilo que pretendemos guardar em ficheiro; `const` indica que o parâmetro não irá ser alterado.
- `size` indica o tamanho em *Bytes* de cada um dos elementos que pretendemos escrever.
- `nElements` indica o número de elementos que pretendemos escrever
- `file` é um descritor que indica o ficheiro onde os dados irão ser colocados.

- RETORNA o número de itens que se conseguiram escrever com sucesso
- ERRO se o número de itens efetivamente escritos for diferente dos pretendidos.

Leitura de blocos de ficheiros binários – função *fread*

SINTAXE
```
size_t fread (const void* ptr, size_t size, size_t nElements, FILE* file);
```

- `ptr` é um apontador para `void` (para qualquer tipo) e contém o endereço de memória onde pretendemos colocar os dados que iremos ler a partir do ficheiro.
- `size` indica o tamanho em *Bytes* de cada um dos elementos que pretendemos ler.
- `nElements` indica o número de elementos que pretendemos ler
- `file` é um descritor que indica o ficheiro de onde os dados irão ser lidos. Este argumento corresponde à variável que recebeu o resultado da função `open`.

- RETORNA o número de itens que se conseguiram ler com sucesso

Exemplos de utilização de ficheiros binários

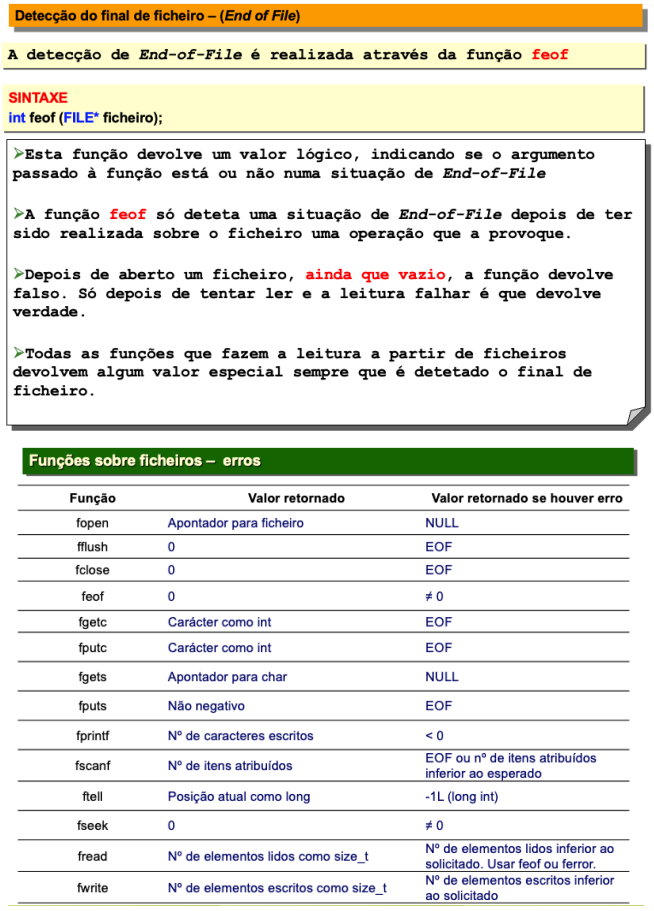
1. Escrever uma função para guardar n valores inteiros num ficheiro binário

```
void insereInteiros(FILE* file, int* vetor, unsigned int n)
{
   if (fwrite(vetor, sizeof(int), n, file) != n) //escreve as n componentes
     fprintf(stderr, "\nNão foram escritos todos os numeros\n");
   fclose(file);
}
```

2. Escrever uma função para abrir um ficheiro e, caso seja possível abri-lo, ler os dados

```
void leFicheiro(FILE* file, char* fileName, int* vetor, unsigned int n)
{ file = fopen(fileName, FILEMODE_rb);
  if (file == NULL)
   fprintf(stderr, "Impossivel abrir o ficheiro.\n");
  else
  {
     fread(vetor, sizeof(int), n, file);
     fclose(file);
  }
}
```

O primeiro argumento das funções fwrite e fread é o endereço do elemento ou conjunto de elementos a escrever no ficheiro. Tratando-se de um vetor, indica-se o seu nome (endereço da primeira componente, equivalente a &vetor[0]).



Detecção do final de ficheiro – (*End of File*)

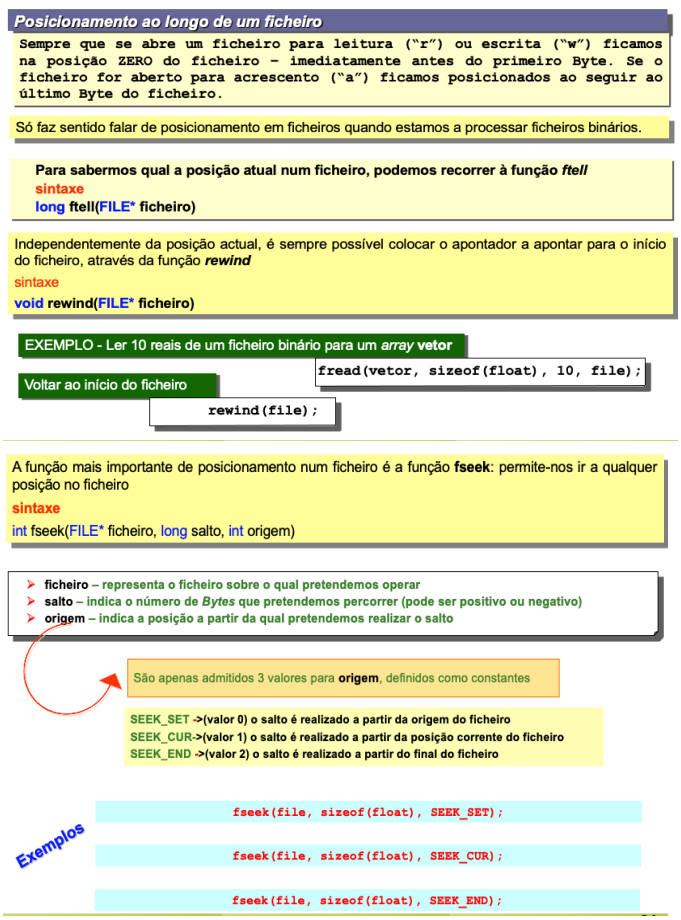
A detecção de *End-of-File* é realizada através da função **feof**

SINTAXE
int feof (FILE* ficheiro);

- Esta função devolve um valor lógico, indicando se o argumento passado à função está ou não numa situação de *End-of-File*
- A função **feof** só deteta uma situação de *End-of-File* depois de ter sido realizada sobre o ficheiro uma operação que a provoque.
- Depois de aberto um ficheiro, **ainda que vazio**, a função devolve falso. Só depois de tentar ler e a leitura falhar é que devolve **verdade**.
- Todas as funções que fazem a leitura a partir de ficheiros devolvem algum valor especial sempre que é detetado o final de ficheiro.

Funções sobre ficheiros – erros

Função	Valor retornado	Valor retornado se houver erro
fopen	Apontador para ficheiro	NULL
fflush	0	EOF
fclose	0	EOF
feof	0	≠ 0
fgetc	Carácter como int	EOF
fputc	Carácter como int	EOF
fgets	Apontador para char	NULL
fputs	Não negativo	EOF
fprintf	Nº de caracteres escritos	< 0
fscanf	Nº de itens atribuídos	EOF ou nº de itens atribuídos inferior ao esperado
ftell	Posição atual como long	-1L (long int)
fseek	0	≠ 0
fread	Nº de elementos lidos como size_t	Nº de elementos lidos inferior ao solicitado. Usar feof ou ferror.
fwrite	Nº de elementos escritos como size_t	Nº de elementos escritos inferior ao solicitado



Posicionamento ao longo de um ficheiro

Sempre que se abre um ficheiro para leitura ("r") ou escrita ("w") ficamos na posição ZERO do ficheiro – imediatamente antes do primeiro Byte. Se o ficheiro for aberto para acrescento ("a") ficamos posicionados ao seguir ao último Byte do ficheiro.

Só faz sentido falar de posicionamento em ficheiros quando estamos a processar ficheiros binários.

Para sabermos qual a posição atual num ficheiro, podemos recorrer à função *ftell*
sintaxe
long ftell(FILE* ficheiro)

Independentemente da posição actual, é sempre possível colocar o apontador a apontar para o início do ficheiro, através da função ***rewind***
sintaxe
void rewind(FILE* ficheiro)

EXEMPLO - Ler 10 reais de um ficheiro binário para um *array* **vetor**

```
fread(vetor, sizeof(float), 10, file);
```

Voltar ao início do ficheiro

```
rewind(file);
```

A função mais importante de posicionamento num ficheiro é a função **fseek**: permite-nos ir a qualquer posição no ficheiro
sintaxe
int fseek(FILE* ficheiro, long salto, int origem)

- **ficheiro** – representa o ficheiro sobre o qual pretendemos operar
- **salto** – indica o número de *Bytes* que pretendemos percorrer (pode ser positivo ou negativo)
- **origem** – indica a posição a partir da qual pretendemos realizar o salto

São apenas admitidos 3 valores para **origem**, definidos como constantes

SEEK_SET ->(valor 0) o salto é realizado a partir da origem do ficheiro
SEEK_CUR->(valor 1) o salto é realizado a partir da posição corrente do ficheiro
SEEK_END ->(valor 2) o salto é realizado a partir do final do ficheiro

Exemplos

```
fseek(file, sizeof(float), SEEK_SET);
fseek(file, sizeof(float), SEEK_CUR);
fseek(file, sizeof(float), SEEK_END);
```

Problemas Propostos

1 - Elabore um programa que aceite repetidamente *strings* do utilizador e as grave num ficheiro. O programa deve terminar a escrita no ficheiro, quando a *string* introduzida for o caracter '.'.

2 - Elabore um programa que leia e mostre no monitor as *strings* contidas no ficheiro gravado pelo programa anterior.

3 - Elabore uma função que permita saber o tamanho (número de *bytes*) de um ficheiro. O nome do ficheiro de que se quer saber o tamanho deve ser pedido ao utilizador.

4 - Complete o programa do exercício 4 da ficha 5 de forma a permitir:

- armazenar os dados num ficheiro;
- ler os dados do ficheiro.

A função que grava os dados em ficheiro deve armazenar apenas as fichas efetivamente preenchidas.