

# ***Estruturas de Dados***

**Engenharia Informática**

1º Ano - 2º Semestre

**Francisco Morgado**

Escola Superior de Tecnologia e Gestão de Viseu

## ***2. Ficheiros***

***Introdução***

***Abertura de ficheiros***

***Modo de texto e modo binário***

***Fecho de ficheiros***

***Processamento de ficheiros de texto***

***Processamento de ficheiros binários***

***Posicionamento ao longo de um ficheiro***

***Aplicação a escrita e leitura de matrizes***

## 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*.

## Abertura de ficheiros

A abertura de um ficheiro permite associar um ficheiro existente num suporte magnético 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**

## 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* alíquotas 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.

## 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)

## 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.

## Modo de texto e modo binário

- Por omissão, 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 carácter *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).



*A representação do carácter New Line é feita em C através da sequência '\n'. Quando a linguagem pretende escrever esta sequência (em ficheiro ou para o ecrã), converte-a nos dois caracteres que nos sistemas MS representam New Line (CR e LF). O mesmo acontece quando o C efetua a leitura de caracteres, convertendo (CR+LF) na sequência '\n' que representa um único carácter.*

A diferença entre o processamento de um ficheiro aberto em modo de texto e aberto em modo binário é que em modo de texto sempre que é encontrada a sequência CR+LF, é lida como se se tratasse de um único carácter, enquanto que no processamento em modo binário é lido um carácter (*Byte*) de cada vez.

A função **fopen** é responsável pela abertura de ficheiros. Se por qualquer razão o ficheiro não puder ser aberto, a função devolve NULL e o programa não aborta, como acontece noutras linguagens. Compete ao programador resolver os eventuais problemas que possam surgir na abertura para que o programa tenha uma sequência adequada.

Se a função **fopen** conseguir abrir um ficheiro com sucesso, cria em memória uma estrutura (do tipo FILE) que representa toda a informação necessária acerca do ficheiro que estiver a processar, devolvendo o endereço em que essa estrutura foi criada. Caso não tenha conseguido abrir o ficheiro, devolve NULL, o endereço ZERO de memória.

## Abertura de ficheiros – Síntese

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

## Modos de abertura de ficheiros

- O modo de abertura de um ficheiro é definido através de uma *string*. Por exemplo: "rb", "wb", "ab", "a+b", etc.
- Contudo qualquer *string* é aceite pelo compilador, originando erros apenas em execução.
- Criação do ficheiro file.h para detectar erros na compilação

```
// file opening modes
#define FILEMODE_r "r"
#define FILEMODE_w "w"
#define FILEMODE_a "a"

#define FILEMODE_rU "r+"
#define FILEMODE_wU "w+"
#define FILEMODE_aU "a+"

#define FILEMODE_rb "rb"
#define FILEMODE_wb "wb"
#define FILEMODE_ab "ab"

#define FILEMODE_rUb "r+b"
#define FILEMODE_wUb "w+b"
#define FILEMODE_aUb "a+b"
```

## 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.

Quando se fecha um ficheiro, os dados que possam existir em memória são escritos fisicamente em disco (*flushed*) e só depois é desligada a ligação entre o ficheiro físico e a variável que o representa.

A função *fflush()* permite ao utilizador gravar fisicamente os dados que ainda possam existir em memória no *buffer* associado ao ficheiro.

### SINTAXE

```
int fflush (FILE* ficheiro);
```

Esta função pode também ser usada para limpar o *buffer* associado ao teclado, através da invocação (*apenas em alguns compiladores!!!*)

```
fflush (stdin);
```

Para fazer o *flush* (gravação) a todos os ficheiros, pode usar-se a função:

```
fflush(NULL);
```

## 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.

## 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.

## Ficheiros standard

Sempre que um programa em C é executado, são automaticamente abertos pelo menos 3 ficheiros *standard*:

- **stdin** – representa o *standard input* e está normalmente associado ao teclado.
- **stdout** – representa o *standard output* e está normalmente associado ao ecrã.
- **stderr** – representa o *standard error* (local para onde são enviadas as mensagens de erro de um programa) e que tipicamente está associado ao ecrã.



## *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**

## 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]**).

## Escrever um programa que utilize as funções `insereInteiros` e `leFicheiro`

```
#include <stdio.h>
#include <stdlib.h> // exit e system
#include "file.h"
const int Max = 20;
void main()
{
    FILE* file; int vn[Max]; unsigned int i, n;
    char fileName[80] = "Dados.DAT";
    file = fopen(fileName, FILEMODE_wb);
    if (file == NULL)
    {
        fprintf(stderr, "\nImpossível criar o ficheiro %s\n", fileName);
        exit(1);
    }
    do
    {
        printf("\nIndique o nº de inteiros a escrever no ficheiro, sem exceder %d ", Max);
        scanf("%d",&n);
    } while (n<1 || n> Max);
    printf("\nDigite os %d números inteiros: ", n);
    for (i=0; i<n; i++)
    {
        printf("\n %d->", i+1);
        scanf("%d", &vn[i]);
        // fwrite(&vn[i], sizeof(int), 1, file); escreveria componente a componente
    }
    insereInteiros (file, vn, n);
    leFicheiro(file, fileName, vn, n);
    system("pause");
}
```

A função `exit (int num)` permite terminar um programa, onde quer que esteja a executar. Em geral devolvem-se números diferentes consoante a razão porque termina. Caso o programa termine sem qualquer problema, devolve-se 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.

## 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);
```



### Aplicação

Escrever um programa que abra o ficheiro Dados.DAT que contém um conjunto de números reais e mostre os valores nele existentes nas posições ímpares do ficheiro (1ª, 3ª, 5ª, ...)

```
#include <stdio.h>
#include <stdlib.h> //porque se utiliza a função exit
#include "file.h"
void main()
{
    FILE* fp = fopen("Dados.DAT", FILEMODE_rb);
    float x;
    if (fp == NULL)
    {
        fprintf(stderr, "\n Impossivel abrir o ficheiro Dados.DAT\n");
        exit(1);
    }
    while (1) //ciclo infinito
    {
        if (fread(&x, sizeof(float), 1, fp) == 0) //falhou a leitura
            break;
        printf("\n%.2f", x);
        fseek(fp, sizeof(float), SEEK_CUR); // avança uma posição
    }
    fclose(fp);
}
```

## 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 ferrord.
fwrite	Nº de elementos escritos como size_t	Nº de elementos escritos inferior ao solicitado

## Aplicação a operações sobre matrizes

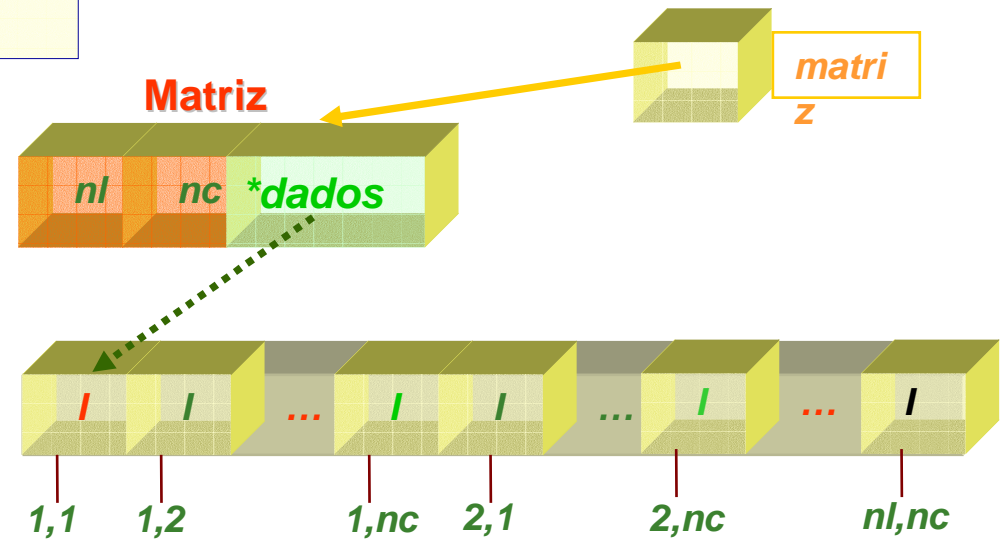
### Elementos introdutórios

### Estrutura

```
typedef int Informacao;  
typedef struct  
{  
    int nl, nc;  
    Informacao* dados;  
} Matriz;
```

### Alocação de uma matriz

```
Matriz* alocaMatriz(int nLinhas, int nColunas)  
{  
    Matriz* matriz; //matriz é um ponteiro para uma variável estruturada do tipo Matriz  
    matriz = (Matriz*)malloc(sizeof(Matriz));  
    //As 2 instruções anteriores são equivalentes à instrução  
    // Matriz* matriz = (Matriz*)malloc(sizeof(Matriz));  
    matriz->nl = nLinhas; // equivalente a (*matriz).nl = nlinhas;  
    matriz->nc = nColunas;  
    matriz->dados = (Informacao*)malloc( nLinhas*nColunas*sizeof(Informacao) );  
    if ( matriz->dados == NULL)  
    {  
        fprintf(stderr, "Memória insuficiente\n");  
        return(NULL); // termina o programa por falta de memória  
    }  
    else  
        return matriz;  
}
```



## Escrita de uma matriz num ficheiro de texto

```
void gravaMatrizFicheiroTexto(Matriz matrizM, char* fileName)
{
    if (matriz == NULL)
        return; // No caso de matriz ser NULL não faz nada!!
    FILE* file = fopen(fileName, "w" );
    if (file == NULL)
    {
        fprintf(stderr, "Problemas na abertura do ficheiro! \n");
        return;
    }
    fprintf(file, " %d \n", matriz->nl);
    fprintf(file, " %d \n", matriz->nc);
    for (int i = 0; i < matriz->nl; i++)
    {
        for (int j = 0; j < matriz->nc; j++)
            fprintf(file, "%d ", matriz->dados[i*matriz->nc + j]);
        fprintf(file, "\n");
    }
    fclose(file);
}
```

## Escrita de uma matriz num ficheiro binário

```
void gravaMatrizFicheiroBinario(Matriz* matriz, char* fileName)
{
    if (matriz == NULL)
        return; // No caso de matriz ser NULL nao faz nada
    FILE* file = fopen(fileName, "wb");
    if (file == NULL)
    {
        fprintf(stderr, "Problemas na abertura do ficheiro! \n");
        return;
    }

    for (int i = 0; i < matriz->nl; i++)
    {
        for (int j = 0; j < matriz->nc; j++);
        fwrite((matriz->dados+(i*matriz->nc + j)), sizeof(Informacao),1,file);
    }

    fclose(file);
}
```



```
fwrite(matriz->dados, sizeof(Informacao), (matriz->nl)*(matriz->nc), file);
```


## Leitura de uma matriz de um ficheiro de texto

```
Matriz* leMatrizFicheiroTexto(char* fileName)
{
    FILE* file = fopen(fileName, "r");
    if (file == NULL)
    {
        fprintf(stderr, "Problemas na abertura do ficheiro! \n");
        exit(1);
    }
    int nl, nc;
    fscanf(file, "%d", &nl);
    fscanf(file, "%d", &nc);
    Matriz* matriz = alocaMatriz(nl, nc);
    for (int i = 0; i < matriz->nl; i++)
        for (int j = 0; j < matriz->nc; j++)
            fscanf(file, "%d", &(matriz->dados[i*matriz->nc + j]));
    fclose(file);
    return matriz;
}
```

## Leitura de uma matriz de um ficheiro binário

```
void leMatrizFicheiroBinario(Matriz* matriz, char* fileName)
{
    if (matriz == NULL)
        return; // No caso de M ser NULL nao faz nada!!
    FILE* file = fopen(fileName, "rb");
    if (file == NULL)
    {
        fprintf(stderr, "Problemas na abertura do ficheiro! \n");
        return;
    }

    for (int i = 0; i < matriz->nl; i++)
    {
        for (int j = 0; j < matriz->nc; j++);
        fread(&(matriz->dados[i*matriz->nc + j]), sizeof(Informacao), 1, file );
    }
    fclose(file);
}
```



```
fread(&(matriz->dados), sizeof(Informacao), (matriz->nl)*(matriz->nc), file);
```

## Operações sobre matrizes: Ler do teclado – Escrever no ecrã – Destruir

```
void leMatrizInt(Matriz* matriz)
{ int i,j;
  for (i=0; i<matriz->nl; i++)
    for (j=0; j<matriz->nc; j++)
    {
      printf("\nQual o elemento %d, %d ? ", i+1,j+1);
      scanf("%d", &matriz->dados[i*matriz->nc+j]);
    }
}
```

```
void escreveMatrizInt(Matriz* matriz)
{ int i,j;
  for (i=0; i<matriz->nl; i++)
  {
    printf("\n");
    for (j=0; j<matriz->nc; j++)
      printf("  %d", matriz->dados[i*matriz->nc+j]);
  }
}
```

```
void destroiMatriz(Matriz* matriz)
{
  if (matriz != NULL) // pode acontecer que matriz seja NULL
  {
    free(matriz->dados);
    free(matriz);
  }
}
```



## Programa principal

```
void main()  
{  
    int vnl, vnc;  
    do  
    {  
        printf("\nEscreva o numero de linhas da matriz, sem exceder %d ", MAX);  
        scanf("%d",&vnl);  
    } while (vnl > MAX);  
    do  
    {  
        printf("\nEscreva o numero de colunas da matriz, sem exceder %d ", MAX);  
        scanf("%d",&vnc);  
    } while (vnc > MAX);  
    Matriz* a = alocaMatriz(vnl, vnc);  
    if(a == NULL)  
    {  
        fprintf(stderr, "\nProblemas ao alocar matriz\n");  
        exit(1);  
    }  
}
```

## Programa principal

```
leMatrizInt(a);  
printf("\nmatriz lida do teclado\n");  
escreveMatrizInt(a);  
  
gravaMatrizFicheiroTexto(a, "matriz.txt");  
a = leMatrizFicheiroTexto("matriz.txt");  
printf("\nMatriz lida do ficheiro de texto\n");  
escreveMatrizInt(a);  
  
gravaMatrizFicheiroBinario(a, "matriz.dat");  
leMatrizFicheiroBinario(a, "matriz.dat");  
printf("\nMatriz lida do ficheiro binário\n");  
escreveMatrizInt(a);  
destroiMatriz(a);  
}
```

## Exercício

Gravar num ficheiro binário a informação relativa às viaturas de uma oficina de reparação auto

```
void gravaDados (FILE* file, Viatura* v, unsigned int num)
{
    // o ficheiro já foi aberto para escrita antes...
    if (fwrite(v, sizeof(Viatura), num, file) != num) //grava os dados das num viaturas
        fprintf(stderr, "\nNão foram gravados todos os dados\n");
    fclose(file);
}
```

Ler do ficheiro a informação previamente gravada

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