



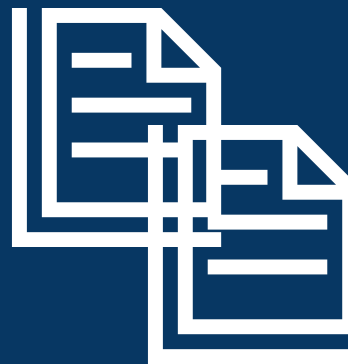
FICHEIROS

CAP ENGEL Diogo Silva
dasilva@academiafa.edu.pt

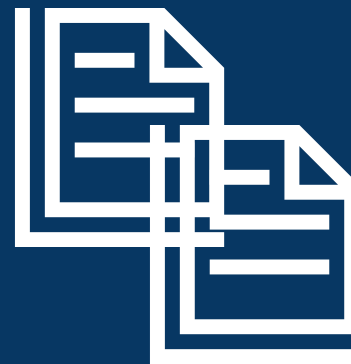
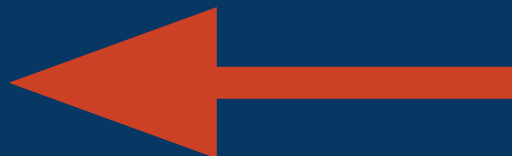
Ficheiros permitem armazenar dados
fora da execução do programa

Programa

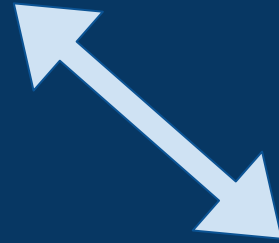
Destino de dados



Fonte de dados



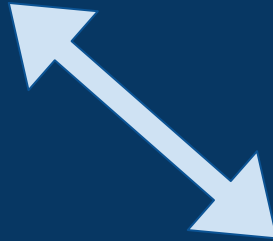
Trabalhar com **ficheiros**



Trabalhar com ***streams***

Uma stream é um conjunto sequencial de caracteres (conjunto de bytes) sem estrutura interna.

1743, Asimov, 1, 20.0, 20.0



49|55|52|51|44|32|65|115|105|109|111|118|44|32|
49|44|32|50|48|46|48|44|32|50|48|46|48|10|

`fopen` **Abertura**

`fgetc, fgets,` **Leitura**
`fscanf, fread`

`fputc, fputs,` **Escrita**
`fprintf, fwrite`

`fclose, fcloseall` **Fecho**

```
#include <stdio.h>
```

```
int main(void) {
```

```
    FILE * fp;
```

```
    fp = fopen("in.txt", "r");
```

```
    fp = fopen("C:/Users/DASilva/Documents/in.txt", "r");
```

```
    if(fp == NULL) {
```

```
        printf("Abertura de ficheiro incorrecta.\n");
```

```
        return 1;
```

```
    }
```

```
    else
```

```
        printf("Ficheiro aberto.")
```

```
    fclose(fp);
```

```
    return 0;
```

```
}
```

```
fp = fopen("in.txt", "r+");
```

Modo de Abertura	Descrição	Leitura	Escrita	Se Fich não Existe	Se Fich 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/Escriver	Sim	Sim	Cria	Altera dados	Início
w+	Ler/Escriver	Sim	Sim	Cria	Recria	Início
a+	Ler/Escriver	Sim	Sim	Cria	Acrescenta dados	Fim

Binário

Acrescentar `b` ao modo de acesso, e.g. `"wb"`, `"rb"`.

Todos os valores são válidos.

Texto

Caracteres da tabela ASCII perceptíveis por nós.

Formatado pelo caracter NewLine (`'\n'`).

```
fp = fopen("in.txt", "r+");
```

```
if(fp == NULL) {
```

fopen retorna um apontador de ficheiro
(FILE *) em caso de sucesso ou NULL.

```
fclose(fp) ;
```

`fclose` fecha o ficheiro apontado por `fp`.
Retorna 0 em caso de sucesso ou a constante EOF (end of file).

`fcloseall` fecha todos os ficheiros abertos pelo programa. Retorna 0 em caso de sucesso em todos os ficheiros ou a constante EOF.

Ficheiros texto

```
int fgetc(FILE *stream)
```

Lê o próximo caracter da `stream` e devolve-o como um `int`, ou `EOF` em caso de fim de ficheiro ou erro.

```
char *fgets(char *s, int size, FILE *stream);
```

Lê no máximo `size-1` caracteres da `stream` e armazena-os em `s`. Leitura termina em `NewLine` ou `EOF`. O caracter terminador de string é adicionado ao final da string. `\n` é adicionado à string se `NewLine` for encontrado na `stream`.

```
int fscanf(FILE *stream, const char *format, ...);
```

Igual a `scanf` mas agora a origem é `stream`.
Devolve `EOF` se detectado final de ficheiro ou o número de parâmetros lidos com sucesso.

Ficheiros texto

```
int fputc(int c, FILE *stream);
```

Escreve `c` na `stream`. Devolve `c` em caso de sucesso ou EOF.

```
int fputs(const char *s, FILE *stream);
```

Escreve `s` na `stream` sem o caracter terminador. Devolve um número não negativo se for bem sucedido ou EOF.

```
int fprintf(FILE *stream, const char *format, ...);
```

Igual a `printf` mas agora o destino é a `stream`. Devolve o número de caracteres escrito na `stream`.

Demo ficheiros texto.

Ficheiros binários

Dados lidos e escritos em blocos de memória. Útil para guardar `struct`.

Não faz sentido o conceito de ler e escrever em linhas, porque não estamos a interpretar os dados como texto.

Ficheiros binários

```
size_t fread(void *ptr, size_t size, size_t nmemb,  
FILE *stream);
```

Lê `nmemb` items de dados (de tamanho `size`) da `stream` e armazena-os na localização apontada por `ptr`.

```
size_t fwrite(const void *ptr, size_t size, size_t  
nmemb, FILE *stream);
```

Escreve `nmemb` items de dados (de tamanho `size`) na `stream`, tendo-os obtido da localização apontada por `ptr`.

Ficheiros binários

`fread`

`fwrite`

Ambas as funções devolvem o número de items lidos/escritos. Esse número é o número de bytes apenas se o tamanho do item for 1. Em caso de erro ou EOF, o valor de retorno é 0 ou um número inferior ao número de items.

Em caso de insucesso, devem ser usadas as funções `feof()` e `ferror()` para saber se ocorreu fim de ficheiro ou erro.

Ficheiros | Posicionamento

```
long ftell(FILE *stream);
```

Devolve posição actual do ficheiro `stream`.

```
void rewind(FILE *stream);
```

Volta ao inicio do ficheiro.

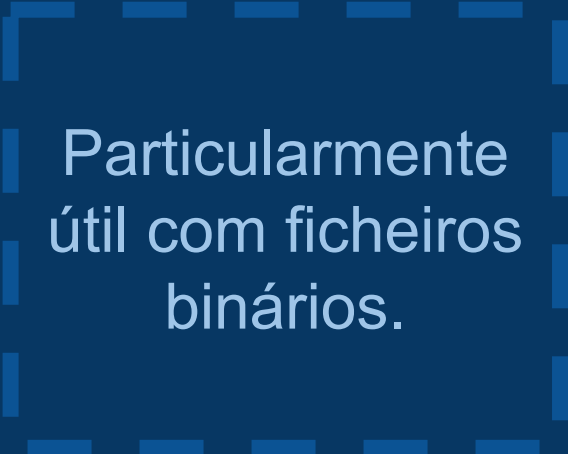
```
int feof(FILE *stream);
```

Devolve inteiro não zero se indicador de fim de ficheiro estiver activo.

Ficheiros | Posicionamento

```
int fseek(FILE *stream, long offset, int whence);
```

Salta para a localização `offset` a partir da posição `whence` no ficheiro `stream`.



Particularmente
útil com ficheiros
binários.

Salto a partir da origem do
ficheiro. `SEEK_SET`

Salto a partir da posição actual
do ficheiro. `SEEK_CUR`

Salto a partir do final do ficheiro.
`SEEK_END`

Demo ficheiros binários.

Crie um programa que lê um ficheiro de texto `alunos.txt` com uma lista de alunos com o formato:

```
Newton 1743
```

```
...
```

```
Asimov 1732
```

O programa deve imprimir no ecrã a informação de todos os alunos no formato:

```
O aluno com numero de CAL 1732 tem o nome  
Asimov.
```

Altere o programa anterior para ler os dados do ficheiro `alunos.txt` para um vetor de registos:

```
typedef struct {  
    char nome[30];  
    int ncal;  
} Aluno;
```

Modifique o programa da cifra por forma a receber a mensagem num ficheiro de texto.

O programa pede ao utilizador o caminho do ficheiro da mensagem e a cifra. Depois, o programa imprime a mensagem cifrada na consola.

O programa dá a opção para gravar a mensagem cifrada noutro ficheiro de texto. Caso esta opção seja escolhida, o programa pede ao utilizador o caminho do ficheiro de saída, e.g.

`C:\\Users\\aluno\\Desktop\\mensagem_secreta.txt`