# Fundamentos de Programação

Ficheiros

P.PORTO
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

# Conteúdo

- Ficheiros de texto
- Ficheiros binários
- Funções `remove` e `rename`

# Stream

- Uma stream é qualquer fonte de **input** ou qualquer destino de **output**.
  - Muitos programas obtêm o seu **input** do teclado e o colocam o seu **output** no ecrã.
  - Normalmente representam ficheiros armazenados em diferentes tipos de média.
  - Podem ser associados com outros dispositivos como network ports e impressoras.

# File pointer

- Aceder a uma stream faz-se com um file pointer cujo tipo é `FILE *`.
  - `FILE` está declarado em `stdio.h`.
- Podemos declarar apontadores para streams:
  - `FILE *fp1, *fp2;`

# stdio.h

- **`stdio.h`** fornece 3 streams que não têm de ser inicializadas nem declaradas para serem utilizadas.

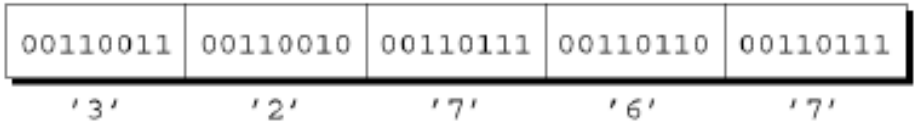
| File Pointer | Stream | Significado |
|---|---|---|
| stdin | Standard input | Teclado |
| stdout | Standard output | Ecrã |
| stderr | Standard error | Ecrã |

# Ficheiro de texto vs Ficheiro binário

- `stdio.h` suporta **ficheiros de texto** e **binários**.
- Os bytes num **ficheiro de texto** representam caracteres que permitem entender o conteúdo.
  - O código fonte de um programa C está num ficheiro de texto.
  - Os ficheiros texto possuem características que não existem nos ficheiros binários:
    - Estão divididos em linhas.
    - Podem conter um marcador de "fim de ficheiro" (**EOF**).
- Num **ficheiro binário** os bytes não representam necessariamente caracteres.
  - Grupos de bytes podem representar outros tipos de dados como inteiros, números de vírgula flutuante, etc.
  - Um programa executável feito em C é armazenado num ficheiro binário.

# Ficheiro de texto vs Ficheiro binário

- Quando escrevemos num ficheiro podemos fazê-lo em formato **texto** ou **binário**.
  - Uma forma de guardar o número **32767** num ficheiro será a de escrever cada um dos seus caracteres **3**, **2**, **7**, **6**, e **7**:




  - A outra opção é guardar em binário (O que utilizaria apenas 2 bytes).

# Abrir ficheiro

```
FILE *fopen(
        const char *filename,
        const char *mode)
```

- `filename` – nome do ficheiro a ser acedido (Pode incluir o caminho).
- `mode` – tipo de operação a executar sobre o ficheiro.
- Retorna:
  - Um file pointer, em caso de sucesso.
  - Caso contrário, NULL.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    printf("Aberto com sucesso %s", FILENAME);

    fclose(fp);

    return 0;
}
```

# Modos

| | String | Significado | Ficheiro |
|---|---|---|---|
| Ficheiros de texto | “r” | Abertura para leitura. | Se o arquivo não existe, `fopen()` retorna `NULL`. |
| | “w” | Abertura para escrita. | Se o arquivo existir, o conteúdo será sobrescrito.<br>Se o arquivo não existir, ele será criado. |
| | “a” | Abertura para acrescentar.<br>Os dados são adicionados ao final do arquivo. | Se o arquivo não existir, ele será criado. |
| | “r+” | Abertura para leitura e escrita. | Se o arquivo não existe, `fopen()` retorna `NULL`. |
| | “w+” | Abertura para leitura e escrita. | Se o arquivo existir, o conteúdo será sobrescrito.<br>Se o arquivo não existir, ele será criado. |
| | “a+” | Abertura para leitura e escrita.<br>Os dados são adicionados ao final do arquivo. | Se o arquivo não existir, ele será criado. |
| Ficheiros binários | “rb” | Abertura para leitura. | Se o arquivo não existe, `fopen()` retorna `NULL`. |
| | “wb” | Abertura para escrita. | Se o arquivo existir, o conteúdo será sobrescrito.<br>Se o arquivo não existir, ele será criado. |
| | “ab” | Abra para anexar no modo binário.<br>Os dados são adicionados ao final do arquivo. | Se o arquivo não existir, ele será criado. |
| | “r+b” ou “rb+” | Abertura para leitura e escrita. | Se o arquivo não existe, `fopen()` retorna `NULL`. |
| | “w+b” ou “wb+” | Abertura para leitura e escrita. | Se o arquivo existir, o conteúdo será sobrescrito.<br>Se o arquivo não existir, ele será criado. |
| | “a+b” ou “ab+” | Abertura para leitura e escrita.<br>Os dados são adicionados ao final do arquivo. | Se o arquivo não existir, ele será criado. |

# Abrir ficheiro

- No Windows é preciso ter cuidado quando a chamada ao **`fopen`** ao incluir o caractere **`\`**.
  - A chamada **`fopen("c:\estgf\test1.dat", "r")`** falha porque **`\t`** é tratado de forma especial.
  - Uma solução é usar **`\\`** em vez de **`\`**:

```
fopen("c:\\folder\\test.dat", "r")
```

  - Outra, é usar **`/`** em vez de **`\`**:

```
fopen("c:/folder/test.dat", "r")
```

# Fechar ficheiro

```
int fclose(FILE *stream)
```

- Retorna:
  - `0`, em caso de sucesso.
  - `EOF`, caso contrário.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    printf(“Aberto com sucesso %s", FILENAME);

    fclose(fp);

    return 0;
}
```

# Ler caractere

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

- Retorna:
  - O próximo caractere em caso de sucesso.
  - **EOF**, caso contrário.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    char ch;

    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    while ((ch = getc(fp)) != EOF) {
        putchar(ch);
    }

    fclose(fp);
    return 0;
}
```

# Ler string

```
char *fgets(char *str,
            int num,
            FILE *stream)
```

- Lê no máximo **num-1** caracteres da **stream** para **str**
  - **str** tem, no final, o **'\0'**.
  - Termina se encontra **'\n'** ou **EOF** .
- Retorna:
  - Um apontador para **str** em caso de sucesso.
  - **NULL**, caso contrário.

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

#define FILENAME "meu-ficheiro.txt"
#define STR_SIZE 25

int main() {
    char str[STR_SIZE];

    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    if (fgets(str, STR_SIZE, fp) != NULL) {
        puts(str);
    }

    fclose(fp);
    return 0;
}
```

# Ler dados formatados

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

- `format` – mesmas regras que o `scanf`.
- Retorna:
  - O número de itens lidos em caso de sucesso.
  - `0`, se o padrão não coincide.
  - `EOF`, em caso de falha.

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

#define FILENAME "meu-ficheiro.txt"
#define STR_SIZE 25

int main() {
    char str[STR_SIZE];
    int num;

    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fscanf(fp, "%s %d", str, &num);
    printf("%s %d", str, num);

    fclose(fp);
    return 0;
}
```

# Escrever caractere

```
int fputc(int character,
          FILE *stream)
```

- Em caso de sucesso escreve o caracter no ficheiro e retorna o caracter escrito.
- Em caso de erro retorna **EOF** e escreve o indicador de erro.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    char letra;

    FILE *fp = fopen(FILENAME, "w");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    for (letra = 65; letra < 91; letra++) {
        fputc(letra, fp);
    }

    fclose(fp);
    return 0;
}
```

# Escrever string

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

- Escreve a string no ficheiro e retorna um valor não negativo.
- Em caso de erro retorna **EOF** e escreve o indicador de erro.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    FILE *fp = fopen(FILENAME, "w");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fputs("test 1", fp);

    fclose(fp);
    return 0;
}
```

# Escrever string

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

- **format** – mesmas regras que o **printf**.
- Retorna:
  - O número de caracteres escritos em caso de sucesso.
  - Um número negativo, caso contrário.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    FILE *fp = fopen(FILENAME, "w");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    if (fprintf(fp, "%s %d", "test", 1) < 0) {
        puts("Aconteceu um erro");
    }

    fclose(fp);
    return 0;
}
```

# Fim de ficheiro

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

- Retorna:
  - Um valor diferente de **0** se está no fim do ficheiro.
  - **0**, se não está no fim do ficheiro.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    char ch;

    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    while ((ch = getc(fp)) && !feof(fp)) {
        putchar(ch);
    }

    fclose(fp);
    return 0;
}
```

# Mover para início do ficheiro

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

- Permite regressar ao início do ficheiro.
- Reinicia o indicador **EOF** e o indicador de erro.

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    char ch;

    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    while ((ch = getc(fp)) != EOF) {
        putchar(ch);
    }

    rewind(fp);

    while ((ch = getc(fp)) != EOF) {
        putchar(ch);
    }

    fclose(fp);
    return 0;
}
```

# Tratamento de erros

```
int ferror(FILE *stream)
```

- Verifica se o indicador de erro da stream tem um valor.
- Retorna um valor diferente de **0** em caso afirmativo.

---

```
void perror(const char *str)
```

- Imprime uma descrição do erro no ficheiro, precedida da string **str** (**str** pode ser NULL).

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

#define FILENAME "meu-ficheiro.txt"

int main() {
    FILE *fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fprintf(fp, "%s %d", "text", 1);
    if (ferror(fp)) {
        perror("error");
    }

    fclose(fp);
    return 0;
}
```

```
error: Bad file descriptor

RUN SUCCESSFUL (total time: 79ms)
```

# Ficheiros binários

- Os **ficheiros binários** têm características que os distinguem dos ficheiros de texto:
  - É possível saltar instantaneamente para qualquer registo, fornecendo acesso aleatório.
  - É possível alterar o conteúdo de um registo em qualquer parte do ficheiro.
- Normalmente têm tempos de acesso mais reduzidos do que os ficheiros de texto.
- A principal desvantagem está relacionada com a portabilidade do código.

# Escrever

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

- **ptr** – apontador para o elemento ou array de elementos a escrever.
- **size** – tamanho de cada elemento.
- **count** – número de elementos a escrever.
- **stream** – apontador para o ficheiro de destino.
- Retorna:
  - O número de elementos escritos.
  - Um número diferente de **count** se aconteceu um erro durante a escrita.

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

#define FILENAME "meu-ficheiro.bin"
#define ALUNOS_MAX 3

typedef struct {
    int numero, media;
} ALUNO;

int main() {
    ALUNO alunos[ALUNOS_MAX] = {
        {.numero = 1, .media = 10},
        {.numero = 2, .media = 15},
        {.numero = 3, .media = 12}
    };

    FILE *fp = fopen(FILENAME, “wb+");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fwrite(alunos, sizeof(ALUNO), ALUNOS_MAX, fp);

    fclose(fp);
    return 0;
}
```

# Ler

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

- **`ptr`** – apontador para um bloco de memória de, pelo menos, **`size*count`** bytes.
- **`size`** – tamanho de cada elemento.
- **`count`** – número de elementos a ler.
- **`stream`** – apontador para o ficheiro de origem.
- Retorna:
  - O número de elementos lidos.
  - Um valor diferente de **`count`** se houve um erro na leitura ou foi atingido o fim do ficheiro.

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

#define FILENAME "meu-ficheiro.bin"
#define ALUNOS_MAX 3

typedef struct {
    int numero, media;
} ALUNO;

int main() {
    int i, num_alunos;
    ALUNO alunos[ALUNOS_MAX];

    FILE *fp = fopen(FILENAME, "rb+");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    num_alunos = fread(alunos, sizeof(ALUNO), ALUNOS_MAX, fp);
    for (i = 0; i < num_alunos; i++) {
        printf("\n%d %d", alunos[i].numero, alunos[i].media);
    }

    fclose(fp);
    return 0;
}
```

# Posicionamento

```
int fseek(FILE *file_ptr,
          long numbytes,
          int origin)
```

- **file_ptr** – apontador para o ficheiro que foi retornado pela chamada ao **fopen**.
- **num_bytes** – Número de bytes a partir de **origin** que nos dará a nova posição.
- **origin** – Uma das seguintes macros:
  - **SEEK_SET**: a partir do início do ficheiro;
  - **SEEK_CUR**: a partir da posição atual;
  - **SEEK_END**: a partir do fim (deslocamento negativo obrigatório).
- Retorna:
  - **0** em caso de sucesso.
  - Outro valor caso contrário.

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

#define FILENAME "meu-ficheiro.bin"
#define ALUNOS_MAX 3

typedef struct {
    int numero, media;
} ALUNO;

int main() {
    ALUNO aluno;

    FILE *fp = fopen(FILENAME, "rb+");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fseek(fp, sizeof(ALUNO) * 2, SEEK_SET);

    fread(&aluno, sizeof (ALUNO), 1, fp);

    printf("\n%d %d", aluno.numero, aluno.media);

    fclose(fp);
    return 0;
}
```

# remove
# rename

```
int rename(const char *old_fn,
           const char *new_fn)
```

- Renomeia o ficheiro **old_fn** para **new_fn**.
- Retorna um valor diferente de **0** se não tive sucesso na operação.

---

```
int remove(const char *fn)
```

- Remove o ficheiro **fn**.
- Retorna um valor diferente de **0** se não tive sucesso na operação.

```c
#include <stdio.h>

#define FILENAME "meu-ficheiro.txt"
#define NEW_FILENAME "meu-ficheiroV2.txt"

int main() {
   int returnValue;

   returnValue = rename(FILENAME, NEW_FILENAME);
   if (returnValue == 0) {
      printf("Renomeado com sucesso.");
   }

   returnValue = remove(NEW_FILENAME);
   if (returnValue == 0) {
      printf("Eliminado com sucesso.");
   }

   return 0;
}
```

# Leitura recomendada

- (**Capítulo 10**) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.