
# Programação I

## Ficheiros de texto e ficheiros binários

Estrela Ferreira Cruz

1

## Objetivos aula

- Ficheiros de texto e ficheiros binários:
  - Conceitos básicos sobre ficheiros;
  - Tipo de ficheiros;
  - Abertura de ficheiros: modos de abertura; função de abertura;
  - Manipulação de ficheiros: funções de leitura e escrita;
  - Fechar ficheiros;
  - Outras funções sobre ficheiros.

2

2

## Ficheiros

- Até agora considerámos que todas as entradas de dados eram feitas a partir do teclado, ou da linha de comando, e que todas as saídas de dados eram feitas para o monitor.
- No final da execução do programa todos os dados manipulados durante a sua execução eram "perdidos".
- Para evitar "perder" esses dados e para aumentarmos as funcionalidades dos programas, vamos aprender armazenar dados em ficheiros e ler os dados de ficheiros.
- Em C, um ficheiro é um conjunto de bytes colocados uns a seguir aos outros de forma sequencial.

3

3

## Ficheiros

Tipos de Ficheiros : Na linguagem C existem dois tipos de ficheiros:

- ficheiros de texto
- ficheiros binários

Ficheiro de texto: Um ficheiro é considerado de texto quando é constituído por caracteres percetíveis para o ser humano: São caracteres da tabela ASCII como algarismos, letras do alfabeto, caracteres de acentuação, pontuação e outros como é o caso do caracter "*New Line*" que, apesar de não ser visível, representa uma mudança de linha.

Ficheiro Binários: Um ficheiro binário pode ser constituído por qualquer caracter existente na tabela ASCII, podendo conter caracteres especiais. Os ficheiros binários não estão organizados por linhas, são escritos em blocos de memória para o disco e lidos em blocos do disco para a memória.

4

4

## Ficheiros

Sempre que pretendemos trabalhar com ficheiros é necessário:

1. Em primeiro lugar abrir o ficheiro, antes de ler ou escrever de/para o ficheiro.
2. Depois de aberto, podemos realizar todas as operações sobre o ficheiro, como seja: ler dados, escrever dados, fazer o posicionamento, etc.
3. Por fim, depois de efetuar as operações necessárias sobre o ficheiro, é necessário fechar o ficheiro.

5

5

## Ficheiros

Regra geral, na linguagem C, as funções de processamento de ficheiros iniciam-se com a letra f (abreviatura de file).

- Em C, para abrir um ficheiro usa-se a função fopen(). Esta função devolve um apontador para o ficheiro.
- Este apontador, aponta para uma estrutura que contém informação referente ao ficheiro como o nome, a posição atual, se foi aberto para escrita ou leitura, etc..
- Este apontador é usado sempre que se acede ao ficheiro, tanto para leituras como para escritas.
- Um apontador para ficheiro é uma variável que tem de ser declarada previamente.

6

6

## Ficheiros

Para declararmos um apontador para ficheiro, usa-se o tipo FILE, pré-definido da linguagem C, que se encontra na biblioteca <stdio.h>.

- Pode ser usado da seguinte forma: FILE *nomeApFicheiro=NULL;
- A função fopen() como o próprio nome indica, abre o ficheiro. Se o ficheiro for aberto com sucesso, associa-o a um apontador para ficheiro. Se não conseguir abrir o ficheiro, a função devolve NULL. A função tem a seguinte sintaxe:

  FILE *fopen(char *nome, char *modo)

- O primeiro argumento da função é uma string com o nome do ficheiro a abrir.
- O segundo argumento é uma string com o modo de abertura do ficheiro.
- O valor de retorno da função fopen() associa o nome físico de um ficheiro à variável do nosso programa. Desta forma, evita-se estar a escrever o nome completo do ficheiro sempre que precisamos de o referenciar.

7

7

## Ficheiros texto

Os principais modos de abertura de um ficheiro texto são apresentados na tabela em baixo:

| Modo | Significado |
|---|---|
| "r" | Abre para leitura (**r**ead). Devolve NULL se o ficheiro não existir ou o utilizador não tiver permissões. |
| "w" | Abre para escrita (**w**rite). É criado um ficheiro com o nome passado na função. Se já existir, o ficheiro é truncado. Se não conseguir abrir, devolve NULL. |
| "a" | Abre para adicionar (**a**ppend). Se o ficheiro não existir cria-o, se já existir, coloca-se no final do ficheiro, de forma a permitir acrescentar dados aos dados já existentes. |
| "r+" | Abre para leitura e escrita (posiciona-se no início do ficheiro) |
| "w+" | Abre para leitura e escrita (o ficheiro é truncado) |
| "a+" | Abre para leitura e escrita (posiciona-se no fim do ficheiro) |

8

8

## Ficheiros

Exemplo:

Para abrir o ficheiro "teste.txt" para escrita podemos usar:

```
FILE *fp=NULL;
fp = fopen("teste.txt","w");
```

Dado que a função pode não conseguir abrir um ficheiro, devolvendo NULL, nesse caso, é necessário verificar o valor de retorno do apontador.

```
FILE * fp=NULL;
fp = fopen("teste.txt","w");
if (fp==NULL) {
    printf("Ocorreu um ERRO ao abrir o ficheiro!\n");
    exit(-1);
 }
```

9

9

## Ficheiros

- Depois de efetuar as operações necessárias sobre o ficheiro, é necessário **fechar o ficheiro**.
- Para **retirar a ligação efetuada na abertura dos ficheiros entre uma variável do nosso programa** e o ficheiro existente no disco, há necessidade de efetuar o fecho do ficheiro.
- Para isso recorre-se à função **fclose()**, cuja sintaxe é a seguinte:

  int fclose(FILE *fp);

- Esta função devolve 0, no caso de existir sucesso no fecho do ficheiro, ou a constante **EOF** no caso de ocorrer um erro.
- Antes do ficheiro ser fechado são gravados todos os dados que possam existir em buffers associados ao ficheiro, sendo também libertada a memória alocada pela função fopen() para a estrutura do tipo FILE.

10

10

## Ficheiros texto

Exemplo de abertura de um ficheiro para leitura:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
FILE *fp=NULL;
char fileName[100];
puts("Introduza o Nome do Ficheiro:");
gets(fileName);
fp = fopen(fileName,"r");
if (fp==NULL){
   printf("Erro ao abrir o ficheiro %s\n", fileName);
}
else {
   printf("Ficheiro %s aberto com sucesso!\n", fileName);
   fclose(fp);
}
}
```

11

11

## Ficheiros

Para escrever informação para ficheiros de texto podemos usar uma das seguintes funções:

```
int fputc(int ch, FILE *fp);

 ou int putc(int ch, FILE *fp);

int fputs(const char *s, FILE *fp);

int fprintf( FILE *fp, const char *format[..]);
```

12

12

## Ficheiros texto

A função int fputc(int ch, FILE *fp);

- A função fputc() permite escrever um caractere num ficheiro previamente aberto para escrita (modo "w" ou "a"), onde fp é o apontador para o ficheiro devolvido pelo fopen() e ch é o caracter a escrever no ficheiro.
- Se a função for bem sucedida devolverá o caracter escrito, se for mal sucedida devolve o caracter EOF.
- EOF é um caracter especial definido na linguagem C que significa fim de ficheiro (End Of File).
- A função putc(int ch, FILE *fp) funciona da mesma forma que a função fputc(int ch, FILE *fp).

13

13

## Ficheiros texto

A função int fputs(const char *str, FILE *fp);

- A função fputs() permite escrever uma string num ficheiro previamente aberto para escrita (modo "w" ou "a"), onde fp é o apontador para o ficheiro devolvido pelo fopen() e str é a string a escrever no ficheiro.
- Nota: a string não pode conter um caractere "new line".
- Se a função for mal sucedida devolve o caractere EOF. Se for bem sucedida devolve um zero.

14

14

## Ficheiros texto

A função

int fprintf( FILE *fp, const char *format[,arg1,..]);

- Permite a escrita formatada de dados em ficheiros de texto.
- O modo de utilização desta função é semelhante à congénere printf() só que, enquanto esta escreve na saída padrão (monitor), a função fprintf() permite escrever para o ficheiro associado ao apontador fp, dado como primeiro argumento da função.
- Se a função fprintf for mal sucedida (se ocorrer um erro na gravação) devolve o caractere EOF. Se for bem sucedida devolve o número de bytes gravados.

15

15

## Ficheiros texto

Exemplo de fprintf: O exemplo seguinte cria um ficheiro de texto e escreve para o ficheiro os números inteiro de 0 a 9, um em cada linha, logo após o cabeçalho.

```
int main() {
   FILE *pf=NULL;
   int i=0;
   fp = fopen(”exemplo.txt", "w");
   if (fp == NULL) {
      printf("Problemas na CRIACAO do ficheiro \n");
      return -1;
   }
   fprintf(fp,”Numeros entre 0 e 9:\n");
   for (i=0;i<10;i++) {
      fprintf(fp,"%d\n",i);
   }
   fclose(fp);
 }
```

16

16

## Ficheiros texto

Para ler informação de um ficheiro de texto podemos usar uma das seguintes funções:

```
int fscanf( FILE *fp, const char *format[...]);

int fgetc(FILE *fp);

 ou int getc(FILE *fp);

char *fgets(char *s, int n, FILE *fp);
```

17

17

## Ficheiros texto

A função int fgetc(FILE *fp):

- Permite ler um caracter de um ficheiro previamente aberto em modo leitura ("r").
- O argumento é um apontador para o ficheiro devolvido pelo fopen().
- A função fgetc() devolve EOF quando o fim do ficheiro é atingido.
- A função getc(FILE *fp) funciona da mesma forma que a função fgetc();
- Assim, para ler um ficheiro até ao fim poderemos usar o código

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

18

18

## Ficheiros texto

**Exemplo:** escrita do conteúdo de um ficheiro para o ecrã.

```
int main(){
FILE *fp=NULL;
char fileName[100];
int ch;
puts("Introduza o nome do ficheiro:");
gets(fileName);
fp = fopen(fileName,"r");
if (fp==NULL){
    printf("Erro ao abrir o ficheiro %s\n", fileName);
    return -1;
}
printf("Coteudo do ficheiro %s :\n", fileName);
ch = getc(fp);
while (ch!=EOF){
       putchar(ch);
       ch = getc(fp);
}
fclose(fp);
return 0;
}
```

19

## Ficheiros texto

**Exercício:** Escreva um programa que implemente o comando *type* do MS-DOS, sendo o nome do ficheiro passado na linha de comando.

```
int main(int argc, char *argv[]) {
FILE *fp=NULL;
int ch;
if (argc!=2) {
     printf("Erro nos argumentos !...");
     exit(-1);     /* Termina o Programa */
}
fp = fopen(argv[1],"r");
if (fp==NULL) {
     printf("Erro ao abrir o ficheiro %s\n",argv[1]);
     return -1;
}
while ((ch=fgetc(fp))!=EOF){
     printf("%c", ch);
}
fclose(fp);
return 0;
}
```

20

20

## Ficheiros texto

A função char *fgets(char *s, int n, FILE *fp);

- A função fgets() permite ler uma string (incluindo o caracter "new line") de um ficheiro de texto.
- O primeiro argumento (s) é a string resultante da leitura. O segundo argumento (n) é o número máximo de caracteres a ler. O terceiro argumento (fp) é o apontador para o ficheiro devolvido pelo fopen() aberto em modo de leitura.
- A função lê texto até encontrar um "new line" ou até um máximo n-1 caracteres. A string resultante é terminada pelo caracter nulo '\0'.
- A função retorna a string lida, caso corra bem. Retorna NULL caso chegue ao fim de ficheiro (encontrou o caracter EOF) ou ocorrer um erro.

21

21

## Ficheiros texto

A função

int fscanf( FILE *fp, const char *format[,arg1,..]);

- A função fscanf () permite ler dados de um ficheiro de texto.
- O modo de utilização desta função é semelhante à congénere scanf() só que enquanto esta lê da entrada padrão (teclado), a função fscanf() permite ler do ficheiro associado ao apontador fp, passado como primeiro argumento para a função.
- A função fscanf() devolve o número de parâmetros que conseguiu ler com sucesso.

22

22

## Ficheiros

Deteção de fim de ficheiro

- Já vimos que há funções que devolvem um caracter especial (EOF) sempre que é detetado o fim do ficheiro.
- A deteção de fim de ficheiro pode ser efetuada usando a função feof() com a seguinte sintaxe:

  int feof(FILE *fich)

- A função feof() devolve o valor 1 quando é atingido o fim do ficheiro e pode ser usada da seguinte forma: while(!feof(fp)){ ….}.
- A função feof() só deteta uma situação de fim de ficheiro depois de ter sido realizada uma operação sobre o ficheiro, sendo que a deteção de fim de ficheiro num ficheiro acabado de abrir devolve Falso, mesmo para um ficheiro vazio.

23

23

## Ficheiros texto

**O Exemplo** seguinte lê um ficheiro texto, linha a linha, com um máximo por linha, de 99 caracteres e imprime o seu conteúdo para o ecrã.

```
int main() {
  FILE *fp=NULL;
  char linha[100], *result=NULL;
  fp = fopen("Teste.txt", "r");     // Abre um ficheiro para LEITURA
  if (fp == NULL) {
        printf(" na abertura do ficheiro \n");
        return -1;
  }
  while (!feof(fp)) {      // Lê uma linha com o '\n' inclusive
        result = fgets(linha, 100, fp);
        if (result) {
              printf("%s", linha);
        }
  }
  fclose(fp);
  return 0;
}
```

24

## Ficheiros texto

**O Exemplo** seguinte lê um ficheiro texto, linha a linha, soma os seus valores e imprime o resultado para o ecrã. O ficheiro contem um inteiro por linha.

```
int main() {
  FILE *fp=NULL;
  int val=0, soma=0, res=0;
  fp = fopen("valores.txt", "r");     // Abre um ficheiro para LEITURA
  if (fp == NULL) {
          printf("Problemas na abertura do ficheiro \n");
          return -1;
  }
   while(!feof(fp)){
         res=fscanf(fp,"%i", &val);
         if (res > 0){
              soma += val;
         }
   }
   printf("Soma=%i\n",soma);
   fclose(fp); return 0;
}
```

25

## Ficheiros Binários

- Os ficheiros binários podem ser constituídos por qualquer caractere existente na tabela ASCII, podendo conter caracteres especiais.
- Nos ficheiros binários a noção de linha não existe, uma vez que os dados não estão organizados por linhas.
- Os dados, neste tipo de ficheiros, são escritos em blocos da memória para o disco e são lidos em blocos do disco para a memória.
- De salientar que as funções que permitem ler ou escrever dados de forma formatada em ficheiros apenas devem ser utilizadas para ficheiros abertos em modo texto.
- Para abrir um ficheiro em modo binário, é necessário acrescentar um b a um dos modos de abertura.

26

26

## Ficheiros Binários

- Modos de abertura de um ficheiro binários são apresentados na tabela seguinte:

| Modo | Significado |
|---|---|
| "rb" | Abre um ficheiro binário em modo de leitura (**r**ead). Devolve NULL se não conseguir abrir o ficheiro. |
| "wb" | Abre um ficheiro binário em modo de escrita (**w**rite). É criado um ficheiro com o nome passado na função. Se já existir o ficheiro é truncado. Se não conseguir abrir devolve NULL. |
| "ab" | Abre um ficheiro binário em modo de acrescentar (**a**ppend). Se o ficheiro não existir cria-o, se já existir, coloca-se no final de forma a permitir a escrita dos dados a partir dos já existentes. |
| r+b | Abre ficheiro binário para leitura (**r**ead) e escrita (write) (posição=inicio). |
| w+b | Cria um ficheiro binário para leitura (**r**ead) e escrita (write) (limpa o ficheiro) |
| a+b | Abre um ficheiro binário para leitura (**r**ead) e escrita (write), em modo de "append" (posição =fim). |

27

27

## Ficheiros Binários

- Nos ficheiros binários a noção de linha não existe, uma vez que os dados não estão organizados por linha.
- Nestes ficheiros os dados são escritos em blocos de memória para o disco e lidos em blocos do disco para a memória.
- Para escrever blocos de dados em ficheiros está definida a função fwrite() que têm a seguinte sintaxe:

```
long fwrite(void *ptr, int size, int num, FILE*fp);
```

- Para ler blocos de dados em ficheiros binários está definida a função fread(), com a seguinte sintaxe:

```
long fread(void *ptr, int size, int num, FILE *fp);
```

28

28

## Ficheiros Binários

A função

long fwrite(void *ptr, int size, int num, FILE *fp);

A função recebe quatro parâmetros:

- Parâmetro 1 - um apontador para a informação a escrever no ficheiro (void *ptr).
- Parâmetro 2 - o tamanho, em número de bytes, de cada elemento a escrever no ficheiro (int size).
- Parâmetro 3 - o número de elementos a escrever no ficheiro (int num).
- Parâmetro 4 - o apontador associado ao ficheiro a que se quer aceder (FILE*fp).

A função retorna o numero de itens que conseguiu escrever com sucesso.

29

29

## Ficheiros Binários

Exemplo de fwrite: Este programa grava 30 elementos do array Notas, com inicio no elemento armazenado no índice 10, para o ficheiro binário.

```
int main() {
    double Notas[100]={23.3,4.2,45.6, …., 45.6,2.3};
    FILE *fp=NULL;
    long result=0;
    // Cria um ficheiro binário para escrita
    fp = fopen("ficheiroTeste.dat", "wb");
    if (fp == NULL)  {
       printf("Problemas na CRIACAO do ficheiro\n");
       return -1;
    }
    //Grava 30 números do vetor a partir do índice 10
    result = fwrite (&Notas[10], sizeof(double), 30, fp);

    printf("Nro de elementos gravados: %ld", result);
    fclose(fp);
    return 0;
}
```

30

30

## Ficheiros Binários

A função

long fread(void *ptr, int size, int num, FILE *fp):

A função recebe quatro parâmetros:

- Parâmetro 1 - um apontador para a região de memória que receberá os dados lidos do ficheiro (void *ptr).
- Parâmetro 2 - o tamanho, em número de bytes, de cada elemento a ler do ficheiro (int size).
- Parâmetro 3 - o número de elementos a ler do ficheiro (int num).
- Parâmetro 4 - o apontador associado ao ficheiro a que se quer aceder (FILE *fp).

A função retorna o numero de itens que conseguiu ler com sucesso.

Nota: retorna o nº de itens e não o numero de bytes!

31

31

## Ficheiros Binários

Exemplo de fread: Este programa lê, de um ficheiro binário, 30 elementos do tipo double, armazena-os em memória no array *val* e em seguida escreve-os no ecrã.

```
int main() {
FILE *fp=NULL;
double val[100];
long result=0;int i=0;
// Abrir ficheiro BINÁRIO para LEITURA
fp = fopen("ficheiroTeste.dat", "rb");
if (fp == NULL) {
     printf("Problemas na abertura do ficheiro\n");
     return -1;
}
result = fread (&val[0], sizeof(double), 30, fp);
printf("Nro de elementos lidos: %ld\n", result);
for (i=0; i<result; i++){
     printf("%f\n", val[i]);
}
fclose(fp);
}
```

32

32

## Ficheiros

- Até aqui, o acesso a ficheiros tem sido feito de forma sequencial, isto é, acede-se a um elemento (leitura ou escrita) e o apontador para o ficheiro fica a apontar para o elemento imediatamente a seguir.
- Quando se quer aceder aos elementos de um ficheiro de forma não sequencial temos de reposicionar o apontador de ficheiro para a posição pretendida usando a função fseek(), que está declarada da seguinte forma:

```
int fseek(FILE *fp, long num_bytes, int ponto_partida);
```

- fp é o apontador para o ficheiro devolvido pela função fopen().
- A função fseek() coloca o apontador de ficheiro na posição num_bytes a partir da origem. A origem, ou ponto de partida, pode ter o valor 0, 1 ou 2 consoante a posição pretendida, estando definidas constantes para cada um dos valores.





## Ficheiros

- Os valores possíveis para origem são:

  0 - SEEK_SET - o salto em num_bytes, é realizado a partir da origem do ficheiro

  1 - SEEK_CUR - o salto em num_bytes, é realizado a partir da posição corrente do ficheiro

  2 - SEEK_END - o salto em num_bytes, é realizado a partir do final do ficheiro

- Esta função fseek() devolve 0 se o movimento dentro do ficheiro foi realizado com sucesso, caso contrário devolve um valor diferente de 0.
- O uso de fseek() não é muito utilizado com ficheiros de texto sendo mais adequado para ficheiros binários.

## Ficheiros Binários

Exemplo uso fseek(): Para ler e escrever para o ecrã o 34º byte do ficheiro binário de nome "teste.dat" podemos usar o código em baixo.

NOTA: O uso do modificador L à constante 34 para forçar o compilador a tratar o valor como um long.

```
int main(){
FILE * fp=NULL;
if((fp=fopen("teste.dat","rb")) == NULL){
      printf("\nErro ao abrir ficheiro!\n");
  }
else{
      fseek(fp,34L,0);
      ch=getc(fp);
      printf("%c",ch);
      fclose(fp);
   }
}
```

35

## Ficheiros

- Para saber qual a posição que ocupa atualmente num ficheiro posso recorrer à função ftell cuja sintaxe é:

  long ftell(FILE *fich)

- Esta função devolve um long pois a dimensão do ficheiro pode ultrapassar o valor máximo representado por um inteiro.
- Independentemente do local onde nos encontramos, é sempre possível voltar a colocar o apontador a apontar para o início do ficheiro através da função rewind(), evitando ter de fechar o ficheiro e voltar a abrir para simplesmente colocar o apontador no início do ficheiro. A sua sintaxe é:

  void rewind(FILE *fich)

36

## Ficheiros

Exercício: Escrever um programa que abra o ficheiro "teste" e indique quantos bytes contem, obtendo este valor através de acesso sequencial.

```
int main( ){
FILE *fp=NULL;
long nBytes=0;
if ((fp = fopen("teste","rb"))==NULL) {
       printf("Impossível Abrir o ficheiro!\n");
       return -1;
}
/* Acesso Sequencial Byte a Byte */
while (fgetc(fp)!=EOF){   /* Ler um char */
       nBytes++;
}
fclose(fp);
printf("Dimensao do Ficheiro = %ld\n",nBytes);
retun 0;
}
```

37

37

## Ficheiros

Exercício: Escrever um programa que abra um ficheiro, cujo nome é recebido na linha de comando e indique quantos bytes este contém. Este valor deve ser obtido através do acesso direto, isto é, sem ter que percorrer todo o ficheiro.

```
int main(int argc, char*argv[]) {
 FILE *fp=NULL;
 if (argc!=2) {
       printf("Sintaxe:\n%s <nome ficheiro>\n",argv[0]);
       return;
 }
 if ((fp = fopen(argv[1],"rb"))==NULL) {
         printf("Impossível Abrir o ficheiro %s\n",argv[1]);
         return -1;
 }
  /* Ir para o fim do ficheiro */
 fseek(fp,0L,SEEK_END);
 printf("Dim. do Ficheiro: %ld\n",ftell(fp));
 fclose(fp);
 return 0;
}
```

38

## Ficheiros

Outras funções sobre ficheiros:

- int fflush(FILE *fich), permite ao programador gravar fisicamente os dados que existem em memória mantendo o ficheiro aberto.
- int fcloseall(), permite fechar todos os ficheiros abertos.
- int flushall(), permite efetuar o flush a todos os ficheiros.
- int remove(char *fileName), permite apagar do sistema o ficheiro "filename".
- int rename(char *oldFileName, char *newFileName), permite mudar o nome do ficheiro "oldFileName" para "newFileName".

39

39

## Ficheiros

Stream

- A linguagem C processa todas as entradas de dados (teclado, rato, etc), saídas de dados (ecrã, impressora, etc) e entrada/saída de dados (discos, disquetes, etc) utilizando streams.
- Uma stream é um conjunto sequencial de caracteres, isto é, um conjunto de bytes sem qualquer estrutura interna, que está sempre ligada a um ficheiro.
- A grande vantagem em utilizar streams para a entrada/saída de dados é o facto de não ser necessário saber que tipo de periférico está a ser utilizado, dizendo-se que é independente do dispositivo.
- Desta forma, não é necessário escrever código diferente para enviar os dados para uma impressora ou para o ecrã, pois os dados são sempre vistos como conjuntos sequenciais de bytes.

40

40

## Ficheiros

O uso de streams **permite associar todas as operações de input e output a um ficheiro**, mesmo que não exista na realidade. Sempre que um programa em C é executado, são abertos no mínimo 5 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**" que é o local para onde devem ser enviadas as mensagens de erro dos programas.
- **Stdaux** - está definido como a porta principal de comunicações, ou seja, a COM1 num PC.
- **Stdprn** - representa o "**standard printer**" e está normalmente associado à **impressora**.

41

41

## Ficheiros

Uma vez que estes ficheiros são abertos automaticamente quando o programa é executado, por exemplo, para escrever uma mensagem na impressora basta fazer:

```
fprintf(stdprn,"Olá Mundo !! \n");
```

Os dados introduzidos pelo teclado não estão associados a um ficheiro, mas é possível processar a leitura desses dados como se viessem de um ficheiro.

Por exemplo, as seguintes instruções são equivalente:

```
scanf("%d %c",&a,&b);
fscanf(stdin,"%d %c", &a, &b);
```

42

42

## Ficheiros

Exemplo:

No programa ao lado ilustra-se a escrita e leitura de estruturas em ficheiros.

```
#include <stdio.h>
#include <string.h>
typedef struct pes {
     char nome[80];
     float alt;
}PESSOA;

void main() {
FILE *fp=NULL;
PESSOA p,r;
fp = fopen("exemplo.dat","wb");
strcpy(p.nome,"jose manuel");
p.alt=1.70;
fwrite(&p,sizeof(PESSOA),1,fp);
fclose(fp);
fp = fopen("exemplo.dat","rb");
fread(&r,sizeof(PESSOA),1,fp);
fclose(fp);
printf("\nNome=%s\nAltura=%f\n",r.nome,r.alt);
}
```

43

## Bibliografia


- Programação Avançada Usando C, António Manuel Adrego da Rocha, ISBN: 978-978-722-546-0.
- Schildt, Herbert: C the complete Reference, McGraw-Hill, 1998.
- Algoritmia e Estruturas de Dados, José Braga de Vasconcelos, João Vidal de Carvalho, ISBN: 989-615-012-5.
- Linguagem C, Luís Manuel Dias Damas, ISBN: 972-722-156-4.
- Elementos de Programação com C - Pedro João Valente D. Guerreiro, 3ª edição, ISBN: 972-722-510-1.
- Introdução à Programação Usando C, António Manuel Adrego da Rocha, ISBN: 972-722-524-1.
- https://bettercodehub.com/


44

44