FICHEIROS binários

As funções de leitura e escrita que permitem acesso direto são **fread** e **fwrite**

fwrite

Faz parte da biblioteca stdio.h e é responsável por escrever um bloco de bytes existentes em memória para um ficheiro aberto em modo binário. Devolve o nº de itens que se conseguirem escrever com sucesso.

Sintaxe:

```
int fwrite (const void *ptr, int size, int n, FILE *fich)
```

em que:

ptr – é um apontador para void (isto é para qualquer tipo) e contém o endereço de memória daquilo que queremos guardar em ficheiro. **const** indica que o parâmetro não será alterado.

size – indica o tamanho em bytes de cada um dos elementos que pretendemos escrever

n – indica o número de elementos que queremos escrever

fich – indica o ficheiro onde serão colocados os dados, este argumento é a variável que recebeu o resultado da função *fopen*

Exemplo (escrita num ficheiro de dados):

```
#include <stdio.h>
#include <stdlib.h> /* por causa da função exit*/
main()
{
	FILE *fp;
	int i, v[10];
	/*ler os dados a partir do teclado*/
	for (i=0;i<10;i++)
	{
		printf("Introduza o %d - esimo N.º: ", i+1);
		scanf("%d",&v[i]);
	}
	/*abrir o ficheiro de dados.dat*/
	fp=fopen("dados.dat","wb");
	if (fp==NULL)
	{
		printf("Impossível criar o ficheiro %s\n","dados.dat");
		exit(1);
	}
	if(fwrite(v,sizeof(int),10,fp)!=10)
		fprintf(stderr, "Não foram escritos todos os elementos!!!\n");
	/* ver ficheiro stderr na página seguinte*/
	fclose(fp);
}
```

Notas:

1- Pretende-se escrever os dados no ficheiro **fp**. São **10** os elementos a escrever, cada um deles com tamanho **sizeof(int)**. Como se trata de um vetor, todos os elementos ocupam espaços contíguos de memória. O endereço do 1º elemento é dado por **&v[0]** que é igual a **v**, pois o nome de um vetor corresponde ao endereço do seu 1.º elemento.

2- A instrução **fwrite(v,sizeof(int),10,fp)** podia ser escrita de várias maneiras:

```
    for (i=0;i<10;i++)
        fwrite(v+i,sizeof(int),1,fp)     /* um de cada vez*/
ou

        fwrite(v, 1,sizeof(int)*10,fp)
```

Analogamente se x fosse uma variável do tipo int, a sua escrita no ficheiro era realizada através da instrução

```
        fwrite(&x, sizeof(int),1,fp);

ou
        fwrite(&x, sizeof(x),1,fp);
```

Ficheiros standard: stdin, stdout,stderr

stdin – representa o standart input e está normalmente associado ao teclado.

stdout – representa o standart output e está normalmente associado ao ecrã.

stderr – representa o standart error (local para onde normalmente devem ser enviadas as mensagens de erro de um programa).Permite enviar mensagens de erro para um local diferente do output de um programa.

As seguintes instruções são equivalentes:

printf("Olá");	fprintf(stdout,"Olá");
puts("Olá");	fputs("Olá", stdout);
scanf("%d",&a);	fscanf(stdin,"%d", &a);

fread

Faz parte da biblioteca stdio.h e é responsável pela leitura para memória de um bloco de bytes existentes num ficheiro aberto em modo binário. Devolve o nº de itens que se conseguirem ler com sucesso.

Sintaxe:

int fread (const void *ptr, int size, int n, FILE *fich)

em que:

ptr – é um apontador para void (isto é para qualquer tipo) e contém o endereço de memória onde queremos co alocar os dados que iremos ler a partir do ficheiro. **const** indica que o parâmetro não será alterado.

size – indica o tamanho em bytes de cada um dos elementos que pretendemos ler.

n – indica o número de elementos que queremos ler.

fich – indica o ficheiro de onde serão lidos os dados, este argumento é a variável que recebeu o resultado da função *fopen.*

Exemplo (leitura a partir de um ficheiro de dados):

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        FILE *fp;
        int i, v[10],n;

        /*abrir o ficheiro de dados.dat*/
        fp=fopen("dados.dat","rb");
        if (fp==NULL)
        {
                printf("Impossível abrir o ficheiro %s\n","dados.dat");
                exit(1);
        }
        n=fread(v,sizeof(int),10,fp);
        if(n!=10)
                fprintf (stderr, "Foram lidos apenas %d elementos!!!\n");

        /*apresentar dados ao utilizador*/
        for (i=0;i<n;i++)
                printf("%2d º Nº: %d\n",i+1,v[i]);
        fclose(fp);
}
```

Deteção de final de ficheiro

A deteção de final de ficheiro é realizada através da função **feof** cuja sintaxe é:

```
int feof(FILE *fich)
```

Esta função devolve um valor lógico, indicando se o argumento passado à função está ou não numa situação de End-of-file.
A função só deteta uma situação de End-of-file depois de ter sido efetuada uma operação sobre o ficheiro que a provoque. A deteção de um ficheiro acabado de abrir devolve falso.

Exemplo (deteção de End-of-file):

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        FILE *fp;
        int i=0, valor,n;
        /*criar ficheiro vazio*/
        fp=fopen("lixo","wb");
        if (fp==NULL)
        {
                printf("Impossível criar o ficheiro %s\n","lixo");
                exit(1);
        }
        fclose(fp);
        /*abrir o ficheiro lixo*/
        if ((fp=fopen("lixo","rb"))==NULL)
        {
                printf("Impossível abrir o ficheiro %s\n","lixo");
                exit(1);
        }
        puts(feof(fp)?"EOF":"NOT EOF");

        /*tentar ler um caráter*/
        fgetc(fp);
        puts(feof(fp)?"EOF":"NOT EOF");
        fclose(fp);
}
```

Nota: depois de aberto o ficheiro (ainda que vazio) a função **feof** devolve falso e só depois de se tentar ler um caráter e falhar é que devolve true.

Exercício:

1- Escreva um programa que grave para um ficheiro de dados (usa o ponteiro fp para tipo de dados FILE) uma string e de seguida fechá-lo. Posteriormente o ficheiro deverá ser reaberto para que sejam contados os carateres introduzidos.

Usa as funções fread e fwrite.

Acesso direto a um ficheiro

Para nos direcionarmos a qualquer posição de um ficheiro sem ter que percorrer todas as posições intermédias usamos o acesso direto, o que só faz sentido quando trabalhamos com ficheiros binários.

Posicionamento ao longo de um ficheiro

Quando manipulamos um ficheiro, existe a todo momento um ponteiro que se desloca de acordo com as leituras e gravações, ou seja, a cada leitura ou gravação este ponteiro avança uma posição. Mas existem algumas funções e procedimentos que permitem a manipulação deste ponteiro.

- Para saber qual a posição atual num ficheiro recorre-se à função **ftell** cuja sintaxe é:

```
long ftell (FILE *fich)
```

Esta função devolve um long e não um int porque a dimensão dos ficheiros pode ultrapassar o valor máximo representado por um inteiro.

- Para posicionar o ponteiro no início do ficheiro (voltar ao início) usa-se a função **rewind** com a sintaxe seguinte:

```
rewind(fich);
```

- Para permitir mover o ponteiro do ficheiro para uma posição preestabelecida usa-se a função **seek**, que só pode ser usado em ficheiros previamente abertos.

  Sintaxe:

```
int fseek (FILE *fich,long salto, int origem)
```

fich – Representa o ficheiro sobre o qual se pretende operar.

salto – (ou offset) indica o nº de bytes que se pretende "andar" (um valor positivo indica que se pretende avançar, um valor negativo indica que se pretende recuar).

origem – Indica o local a partir do qual queremos realizar o salto no ficheiro. São apenas admitidos 3 valores que são definidos como constantes:

Constante	Valor	Significado
SEEK_SET	0	O salto é realizado a partir da origem do ficheiro
SEEK_CUR	1	O salto é realizado a partir da posição corrente do ficheiro
SEEK_END	2	O salto é realizado a partir do final do ficheiro

A função **seek** devolve 0 se o movimento dentro do ficheiro realizado com sucesso. Caso contrário devolve um valor diferente de zero.

Exemplo (posicionamento ao longo de um ficheiro):

```
#include <stdio.h>
#include <stdlib.h>
main() {
        FILE *fp;
        int i, v[10];
        /*abrir o ficheiro de dados.dat*/
        fp=fopen("dados.dat","rb");
        if (fp==NULL)
        {
                printf("Impossível abrir o ficheiro %s\n","dados.dat");
                exit(1);
        }
        /*ir para o fim do ficheiro dados.dat*/
        fseek(fp, 10, SEEK_END);     /* OU fseek(fp, 10, 2);   */
        printf("Dimensão do ficheiro: %d\n", ftell(fp));
        fclose(fp);
}
```

Outras funções de manipulação de ficheiros:

rename – Muda o nome do ficheiro externo para novo nome.

Sintaxe:

```
int rename(const char *old_filename, const char *new_filename)
```

Exemplo:

```
#include <stdio.h>

int main ()
{
   int ret;
   char oldname[ ] = "file.txt";
   char newname[ ] = "newfile.txt";
   ret = rename(oldname, newname);
   if(ret == 0)
      printf("File renamed successfully");
   else
      printf("Error: unable to rename the file");
}
```

remove – Elimina o ficheiro do disco associado à variável indicada.

Sintaxe:

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

Exemplo:

```
…
int ret;
FILE *fp;
char filename[ ] = "file.txt";
…
ret=remove(filename);
```

```
//continuação do exemplo
if(ret == 0)
        printf ("File deleted successfully");
else
   printf ("Error: unable to delete the file"); …
```

perror – imprime uma mensagem descritiva do erro ocorrido

Sintaxe:

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

Exemplo:

```
#include <stdio.h>
int main ()
{
   FILE *fp;
   rename("file.txt", "newfile.txt");

   /* tentativa de abertura do ficheiro com o nome que tinha antes */
   fp = fopen("file.txt", "r");
   if( fp == NULL )
   {
      perror("Error: ");
      return(-1);
   }
   fclose(fp);
}
```

Exercícios

2. Escreve um programa que solicite ao utilizador 10 números reais e os armazene em ficheiro. Em seguida deves solicitar ao utilizador um nº entre 1 e 10 e mostrar o valor foi introduzido nessa ordem. Deve também mostrar o 1.º e o último elemento do ficheiro.

Exemplo:

```
Introd. O 1-esimo N.º real: 1.2
Introd. O 2-esimo N.º real: 2.3
Introd. O 3-esimo N.º real: 3.4
Introd. O 4-esimo N.º real: 4.5
Introd. O 5-esimo N.º real: 5.6
Introd. O 6-esimo N.º real: 6.7
Introd. O 7-esimo N.º real: 7.8
Introd. O 8-esimo N.º real: 8.9
Introd. O 9-esimo N.º real: 9.1
Introd. O 10-esimo N.º real: 10.2

Qual a ordem do Número que pretende ver 1..10: 4
O 4.º valor introduzido foi 4.5
O 1.º valor introduzido foi 1.2
O último valor introduzido foi 10.2
```

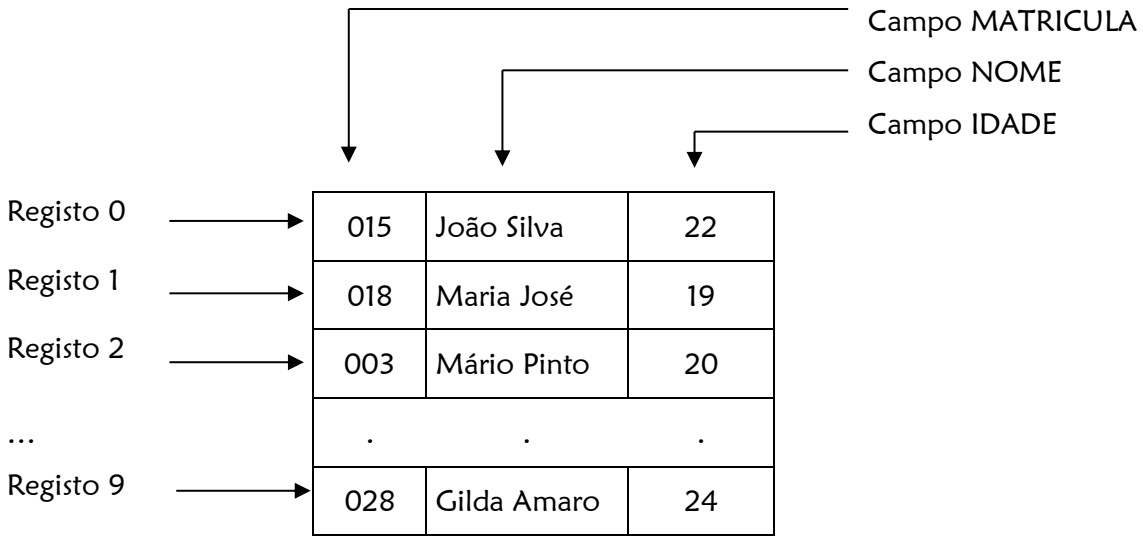
3. Implementa um programa que abra o ficheiro criado no exercício anterior e mostre os valores nele presente apenas nas posições ímpares do ficheiro (isto é, os elementos com índice 1,3,5,…).

Ficheiros de estruturas

O processamento de estruturas armazenadas em ficheiros é sempre realizado recorrendo a ficheiros abertos em modo binário.

- As estruturas são, em geral, lidas e escritas através das funções **fread** e **fwrite**, respetivamente.
- O posicionamento ao longo dos registos ao longo dos registos de um ficheiro pode ser realizado através das funções **rewind** e **fseek**.
- Para se saber a posição atual onde nos encontramos num ficheiro, isto é, qual o nº de bytes para que aponta, na posição atual, o apontador para o ficheiro, usa-se a função ***ftell.***

Estrutura do FICHEIRO ALUNOS, após a sua declaração e utilização, supondo que existam 10 (dez) registos:



Exclusão de registos

No C (assim como na generalidade das linguagens) não existe um comando específica que permita apagar registos no início, ou no meio do ficheiro, recorrendo-se para isso a um conjunto de instruções que permitem a sua eliminação:

1ª Etapa: EXCLUSÃO LÓGICA – Ao especificarmos que um determinado registo deva ser apagado, devemos substituí-lo por um registo em branco, marcando-o, isto é, alterar o conteúdo dos seus campos numéricos para 0 (zero) e os seus campos tipo string para " " (espaço em branco);

2ª Etapa: EXCLUSÃO FÍSICA – Após definirmos todos os registos que deverão ser apagados, devemos copiar todos os registos não marcados para serem apagados para um outro FICHEIRO temporário. Feito isto, apagamos o FICHEIRO original e renomeamos o FICHEIRO temporário com o nome do FICHEIRO original, de modo a que fiquemos apenas com os registos que não pretendíamos eliminar.

EXERCÍCIOS

4. Escreve um programa que crie um novo FICHEIRO chamado:

a) AGENDA.DAT contendo os campos NOME e FONE, e grava no mesmo 5 registos lidos pelo teclado;

b) PESSOAS.DAT contendo os campos NOME e IDADE, e grava no mesmo um nº indeterminado de registos lidos pelo teclado (o fim da introdução é definida pela leitura do nome '0');

5. Escreve um programa que:

a) Abra o FICHEIRO AGENDA.DAT criado no exercício anterior e exiba os 5 registos no monitor do computador, utilizando: ***for (i=0 ;i<5;i++);***

b) Permita acrescentar registos ao ficheiro AGENDA.DAT.

6. Escreve um programa que abra o FICHEIRO PESSOAS.DAT criado anteriormente e exiba todos os seus registos no monitor do computador (supondo não ser conhecida a quantidade de registos).

7. Escreve um programa que abra o FICHEIRO PESSOAS e exiba todos os nomes com idade superior ou igual a 15:

Nota: Deve-se aceder a todos os registos e selecionar aqueles cuja idade seja maior ou igual a 15.

8. Escreve um programa que:

a) faça a inclusão de registos num FICHEIRO chamado ALUNOS. Se o FICHEIRO não existir, o programa deve criar um FICHEIRO vazio e proceder à inclusão (consulta de novo a página 4). A leitura da matrícula 0 (zero) indica fim dos dados.

"Lay-out" do FICHEIRO: **MATRICULA**
NOME
MEDIA

Nota: Deve-se mover o ponteiro para a posição posterior ao último registo e então efetuar a gravação dos dados lidos pelo teclado.

b) Permita pesquisa de registos do ficheiro alunos, pelo campo matrícula e sua impressão no ecrã. A leitura da matrícula 0 (zero) indica o fim das pesquisas.

9. Escreve um programa que altere a média de um aluno no FICHEIRO ALUNOS.

Nota: Deve-se solicitar a digitação da matrícula do aluno que se deseja alterar e percorrer todo o FICHEIRO comparando a matrícula de cada aluno com a matrícula digitada. Ao encontrar, exibe-se os dados atuais e solicita-se a digitação da nova média.

10. Escreve um programa que nos permita excluir registos do FICHEIRO ALUNO