Ficha de Trabalho n.º 2

Ficheiros de Texto

Para declarar e abrir ficheiros usa-se a função

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

- devolve um ponteiro para o ficheiro se o abrir correctamente ou NULL caso falhe a abertura
- tem como argumentos o nome do ficheiro e o modo de abertura:
  - **r** (*read*) Abre o ficheiro para ler. O ponteiro aponta para o início do ficheiro.
  - **w** (*write*) Abre o ficheiro para escrever. Se existir apaga-o, se não cria-o.
  - **a** (*append*) Abre o ficheiro para "acrescentar". Se não existir cria-o, se existir coloca o ponteiro no fim e não apaga.
  - **r+**, **w+** Abre para escrita e leitura. Com w+ apaga o conteúdo e cria se não existir. O ponteiro fica no início.
  - **a+** Abre o ficheiro para ler ou escrever. Se não existir cria-o, se existir coloca o ponteiro no fim. Permite mover o ponteiro para trás e fazer leituras.
  - **b** (*binary*) Abre para ler ou escrever um ficheiro binário.

  É possível combinar os diferentes modos: "rb", "wb", "ab", "r+b", "w+b", "a+b", "ra", "r+w",...

Para fechar ficheiros usa-se a função

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

- permite fechar qualquer ficheiro. Todos os ficheiros que se abrem devem ser fechados para garantir a sua integridade.
- recebe um ponteiro para o ficheiro e devolve 0 se a operação correu bem, ou a constante EOF em caso de erro.

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *f;
  f = fopen("ficheiro.txt","r");//fopen(nome do ficheiro, modo de abertura)

  if(f == NULL)
  {
    printf("Erro: o ficheiro nao existe ou nao pode ser lido...\n");
    return -1;
  }
  else
    printf("o ficheiro existe!\n",f);

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

Instruções de escrita em ficheiros

- **fprintf**(): semelhante à **printf**() mas recebe mais um parâmetro, que é o ponteiro para o ficheiro.

```
fprintf(f,"texto a escrever no ficheiro f");
```

- **fputs():** semelhante à **puts() –** permite escrever uma string sem formatação – mas recebe mais um parâmetro que é o ficheiro onde a string deverá ser escrita.
- **fputc**(): semelhante à putc() – permite escrever apenas um caractere – mas recebe também um parâmetro que é o ficheiro onde deverá ser escrito o caractere.

Instruções de leitura de ficheiros

- **fscanf**(): semelhante à do **scanf**(), mas tem mais um parâmetro que é o ficheiro origem.

```
fscanf(f,"%i",&n);
```

- **fgets**(): permite ler uma string com mais do que uma palavra de um ficheiro de texto.
  É semelhante à **gets**(), mas tem mais um parâmetro que é o ficheiro origem e outro que é o tamanho máximo a ler.
  fgets() lê até ao tamanho máximo, ao fim da linha ou o fim do ficheiro e devolve um ponteiro para a string de dados ou NULL se a leitura falhou.
  (Por permitir limitar o número máximo de caracteres a ler, esta função, ao contrário da gets() que permite *overflow,* é segura)
- **fgetc**() lê um caractere de um ficheiro. Devolve EOF em caso de erro.

```
c = fgetc(f);
```

Outras funções para percorrer ficheiros

- **rewind**() permite voltar ao início do ficheiro

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

- **fseek**() permite mover o ponteiro para qualquer posição do ficheiro:

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

  - devolve 0 se o movimento foi efectuado com sucesso, de outra forma devolve o offset actual do ponteiro
  - **f** é o ponteiro para o ficheiro
  - **offset** é o deslocamento a efetuar, em bytes
  - **whence** é o ponto de referência, a posição a partir de onde se move o ponteiro:
    - **SEEK_SET** – move o ponteiro para o início do ficheiro
    - **SEEK_CUR** – mantém o ponteiro na posição actual
    - **SEEK_END** – move o ponteiro para o fim do ficheiro

```
fseek(f,0,SEEK_SET);//Voltar a posicionar o ponteiro no inicio
fseek(f,0,SEEK_END);//Colocar o ponteiro no fim do ficheiro
fseek(f,3,SEEK_SET);//Colocar o ponteiro a 3 bytes do inicio do ficheiro
fseek(f,-4,SEEK_END);//Colocar o ponteiro 4 bytes antes do fim do ficheiro
fseek(f,3,SEEK_CUR);//Colocar o ponteiro 3 bytes a seguir a posição actual
```

Fim de um ficheiro

- Uma leitura (por exemplo com fscanf) devolve o valor **EOF**;

```
char c = fgetc(f);
while(c != EOF)
{
      printf("%c",c);
      c = fgetc(f);
}
```

- Uma leitura com fgets devolve **NULL**, devendo corresponder ao fim do ficheiro;

```
while(fgets(texto,15,f) != NULL)
   puts(texto);
```

- função **feof**() detecta a ocorrência do fim do ficheiro:

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

  - Devolve 0 se o ponteiro não aponta para o fim do ficheiro e >0 caso contrário
  - O final do ficheiro só é detectado **depois** de uma leitura ter gerado um erro. Assim, a última leitura feita é sempre inválida, sendo necessário ter cuidado.

```
while(!feof(f))
      printf("%c",getc(f));
```

Para cada uma das questões seguintes crie os subprogramas que considerar necessários, testando-os no *main*. Pode atribuir os nomes que considerar adequados, bastando identificar a questão a que se referem.

1. Escreva três subprogramas que executem as seguintes tarefas:
- criar um ficheiro com o nome e conteúdo indicados pelo utilizador;
- acrescentar informação, fornecida por um utilizador, a um ficheiro;
- escrever na consola o conteúdo de um ficheiro.

Complete o programa de modo que seja possível testar os subprogramas.

2. Elabore subprogramas que contabilizem
- **a.** o número de linhas;
- **b.** o número de palavras;
- **c.** o número de caracteres;
- **d.** o número de ocorrências de um caractere indicado pelo utilizador;

existentes num determinado ficheiro de texto indicado pelo utilizador.

3. Altere os subprogramas anteriores de forma que seja escrito, no final do ficheiro, a informação de cada uma das alíneas, assim como a hora em que foram registadas.
NOTA: procure na biblioteca <time.h> as funções time(...), ctime(...).

4. Elabore os subprogramas que considerar necessários para criar ficheiros de *n* números naturais (100, 500, 1000) ordenados por ordem crescente, por ordem decrescente e aleatoriamente. O nome do ficheiro deve indicar o número de elementos e modo como estão ordenados. Por exemplo, o ficheiro *aleat500.txt* contém os números naturais de 1 a 500 ordenados aleatoriamente.

5. Desenvolva um subprograma que escreva num ficheiro os primeiros 50 números do ficheiro *aleat100.txt*, os 100 primeiros números do ficheiro *aleat500.txt* e os primeiros 500 números do ficheiro *aleat1000.txt*.
Analise o ficheiro obtido e separe em dois novos ficheiros os números pares (em *pares.txt*) dos números ímpares (em *impares.txt*).
Averigue se há mais números pares ou números ímpares no primeiro ficheiro criado.

6. Desenvolva um programa que faça a gestão dos alunos inscritos numa turma. Para cada aluno deve conhecer-se o nome e o número.
O programa deve permitir:

- Listar os alunos inscritos;
- Inscrever um aluno;
- Remover um aluno da lista.

A lista de alunos deve manter-se entre as várias execuções do programa.

Ficheiros Binários

Tipos de ficheiros

- **texto**
  - cada linha é composta por caracteres legíveis;
  - permite apenas acesso sequencial ao conteúdo;
- **binário**:
  - composto por sequências de bytes ilegíveis, caracteres que não são percetíveis em editores de texto, são copiados directamente das variáveis na memória;
  - permite acesso sequencial e acesso directo aos valores.

A criação, a abertura e o fecho de ficheiros binários são programados da mesma forma que os ficheiros de texto mas o **modo de abertura deve incluir "b"**.

```
#include <stdio.h>

#define FICHB "dados.dat"

int main(int argc, char *argv[])
{
    FILE *f;
    f = fopen(FICHB,"rb");

    if(f == NULL)
    {
        printf("Erro: o ficheiro %s não pode ser lido...\n",FICHB);
        return -1;
    }
    fclose(f);
    return 0;
  }
```

Instruções de escrita em ficheiros binários não usam escrita formatada

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

- devolve o número de elementos (blocos) escritos com sucesso
  caso devolva 0, nada foi escrito, deve ter ocorrido um erro
- `ptr`: ponteiro com endereço de memória do que se pretende escrever
- `size`: tamanho (em bytes) de cada elemento (bloco) a escrever
- `n`: número de elementos (blocos) a escrever
- `f`: ponteiro para o ficheiro onde escrever.

Instruções de leitura de ficheiros binários

```
int fread(const void *ptr,int size,int n,FILE *f);
```

- devolve o número de elementos (blocos) lidos com sucesso
- `ptr, size, n` e `f`: têm o mesmo significado que em `fwrite`.

Para saber o tamanho dos elementos a ler ou a escrever pode usar-se a função **sizeof**(): `sizeof(int)`, `sizeof(char[20])`,...

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *f;
    char c1 = 'a', c2 = 'b';
    f = fopen("dados.dat","w+b");

    if(f==NULL)
    {
        printf("Erro ao abrir o ficheiro dados.dat.");
        exit(-1);
    }

    fwrite(&c1, sizeof(char), 1, f);//escrever 1 bloco do tamanho de
char, no ficheiro f
    rewind(f);
    fread(&c2, sizeof(c1), 1, f); //ler para c2, de f, 1 bloco do
tamanho de c1
    printf("caractere lido: %c\n",c2);
    fclose(f);
    return 0;
}
```

Para cada uma das questões seguintes crie os subprogramas que considerar necessários, testando-os no *main*. Pode atribuir os nomes que considerar adequados, bastando identificar a questão a que se referem.

7. Para averiguar as diferenças entre ficheiros de texto e ficheiros binários, crie subprogramas para cada uma das seguintes alíneas. Todas as alíneas devem ser testadas no *main*.

- **a.** Crie uma tabela com vários nomes inseridos pelo utilizador;
- **b.** Guarde os nomes da tabela anterior num ficheiro de texto chamado *nomes.txt*.
- **c.** Leia o conteúdo do ficheiro anterior e escreva-o no terminal;
- **d.** Guarde em dois ficheiros binários diferentes, de duas formas diferentes o conteúdo da tabela de a.:
  - i. Num ficheiro binário chamado *tabelaNomes.dat*, guarde a tabela de uma só vez:
  - ii. Num ficheiro binário chamado *nomesBinario.dat*, guarde cada um dos nomes da tabela.
- **e.** Leia o conteúdo de cada um dos ficheiros binários anteriores e escreva no terminal.

Abra os três ficheiros e compare o conteúdo.

8. Refaça o exercício 6: usando ficheiros binários, desenvolva um programa que faça a gestão dos alunos inscritos numa turma. Para cada aluno deve conhecer-se o nome e o número.
O programa deve permitir:

- Listar os alunos inscritos;
- Inscrever um aluno;
- Remover um aluno da lista.

A lista de alunos deve manter-se entre as várias execuções do programa.

9. Pretende-se guardar num ficheiro binário chamado *temperaturas.dat* os dados recolhidos por um termómetro digital automático que regista a temperatura de 15 em 15 minutos:

- Data do tipo long long int (exemplos 20200102, 20200227,...)
- Hora do tipo float (exemplos 00.15, 12.50, 23.45,...)
- Temperatura do tipo float (exemplos 12.3, 19.8,...)

Na impossibilidade de enviar os dados, estes são inseridos manualmente.
Crie subprogramas que permitam realizar as seguintes tarefas e teste-os no *main*:

a. Acrescente ao ficheiro os dados, inseridos pelo utilizador, sabendo que deverá inserir a data, a hora e a temperatura;

b. Calcule a amplitude térmica (temperatura máxima – temperatura mínima) durante um dia. A data deve ser passada como parâmetro, e a função devolve a amplitude.

c. Tal como a amplitude térmica, calcule a média das temperaturas registadas durante um dia específico.