Ficheiros de texto

Ficheiros e organização em disco rígido

Motivação

- Quando há necessidade de guardar informação de uma forma permanente, mesmo quando o computador é desligado

Ficheiro

- Unidade lógica de armazenamento de dados

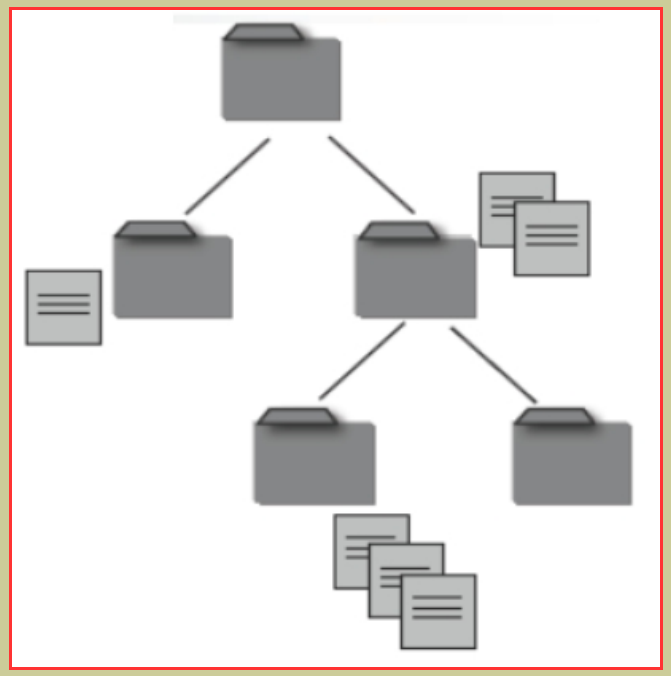
Discos rígidos

- Principais dispositivos de memória não-volátil onde são guardados os ficheiros

Sistema de ficheiros

- Árvore n-ária (invertida) que representa, em termos conceptuais, como todo o disco rígido (ou qualquer unidade de armazenamento) está organizado
- O sistema de ficheiros é composto por diretorias/pastas e ficheiros

Sistema de ficheiros em disco rígido (representação)



Formatos de ficheiros (exemplos)

- .doc (MS Word)
- .c (código em linguagem C)
- .html (hipertexto – internet)
- .exe (executável – código máquina)

Taxonomia de tipos de dados

Simples

- Numéricos
  - int (inteiros)
  - float (reais)
  - char (carateres)
- Apontadores
  - *
- Enumerados
  - enum

Compostos

- Definidos pelo utilizador
  - array (cadeias e tabelas)
  - struct (registos/estruturas)
  - **FILE** (ficheiros)

Tipo de dados FILE

Declaração de uma variável do tipo FILE

- as variáveis do tipo FILE estão associadas a ficheiros em disco rígido (isto é, a dados armazenados em memória não-volátil)
  - as variáveis dos restantes tipos estão associadas a dados armazenados na RAM (isto é, a dados armazenados em memória volátil)
- variável do tipo FILE e ficheiro que lhe está associado
  - a variável só existe durante a execução do programa onde está definida (declarada)
  - o ficheiro pode existir em disco antes da execução do programa e manter a sua existência em disco após o término da execução do programa

Sintaxe

```
FILE *nome;
```

em que

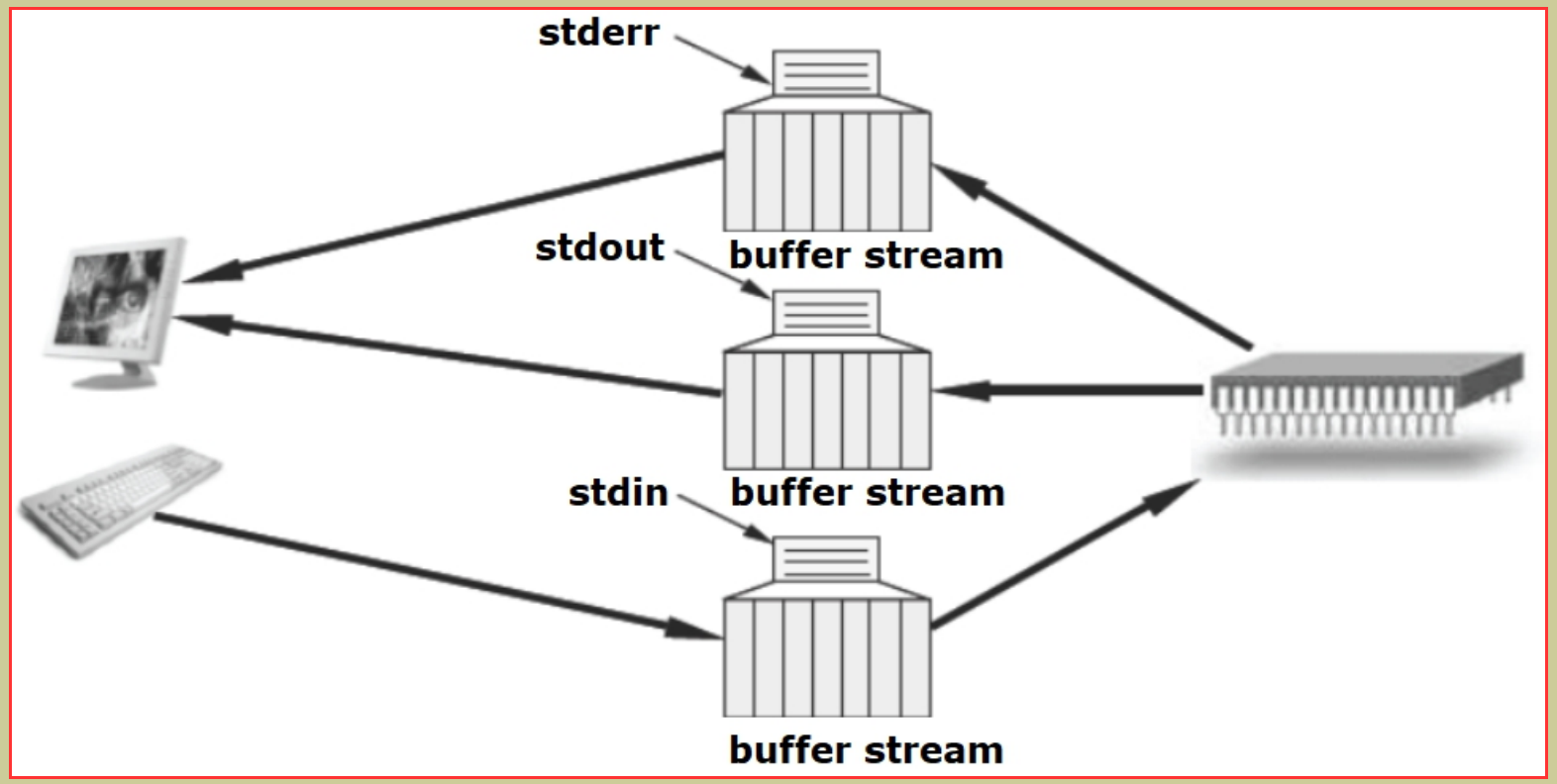
- **nome**: identificador da variável a associar a um ficheiro (nome lógico do ficheiro)
- ***:** indicador de que a variável declarada é do tipo apontadora
- **FILE**: palavra reservada associada ao tipo de dados associado a ficheiros

Ficheiros padrão (standard)

Existem 3 ficheiros padrão

- ficheiro de entrada padrão (**stdin**):
  - associada ao ficheiro físico que é, por defeito, o teclado
- ficheiro de saída padrão (**stdout**):
  - associada ao ficheiro físico que é, por defeito, o monitor
- ficheiro de erro padrão (**stderr**):
  - associada ao ficheiro físico que é, por defeito, o monitor

Representação gráfica

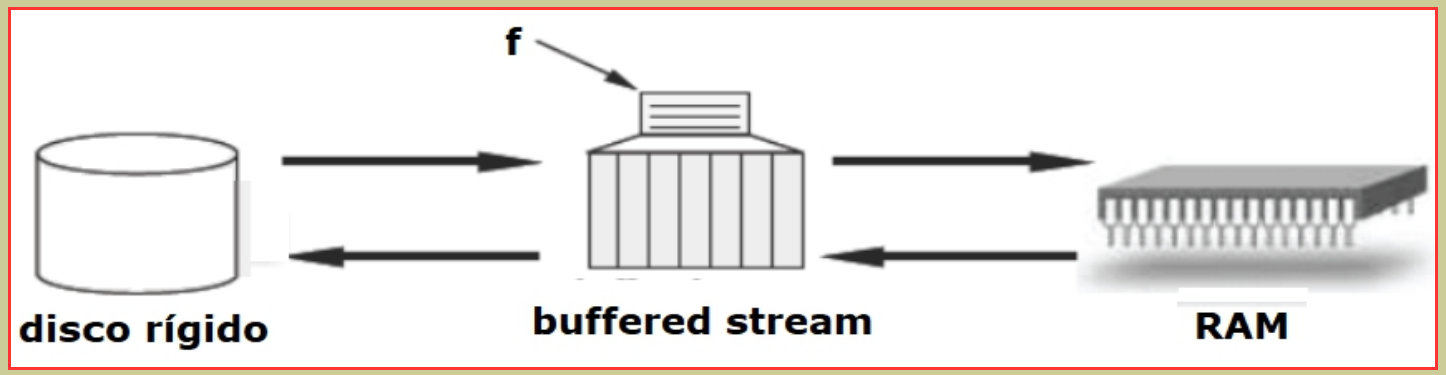


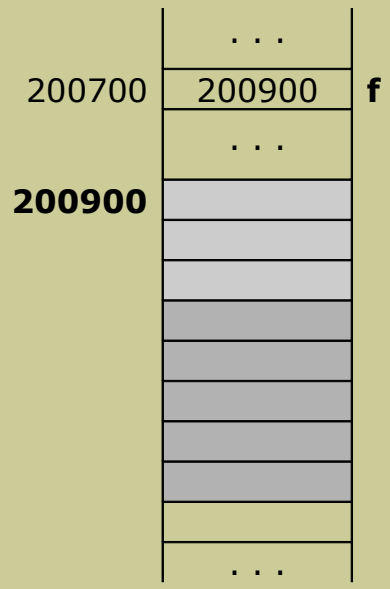
Ficheiros lógicos e ficheiros físicos

Associação entre ficheiro lógico e ficheiro físico

- Ao associar uma variável do tipo **FILE** a um ficheiro físico em disco (usando uma função para tal), esta conterá o nome lógico de ficheiro (ficheiro lógico)

Representação gráfica





FILE

buffer (entrepósito)

Porquê ficheiros lógicos e físicos

- Independência relativamente aos dispositivos físicos
  - A qualquer dispositivo físico de E/S ou ficheiro físico é associado um ficheiro lógico do mesmo tipo (FILE)
  - A manipulação dos dispositivos físicos de E/S é feita de forma uniforme para todos eles
  - Enquanto que:
    - uma variável do tipo FILE só existe durante a execução do programa,
    - o ficheiro ou o dispositivo físico que lhe está associado pode existir antes e manter a sua existência em disco após o término da sua execução
- Redirecionamento do fluxo de E/S
  - É possível redirecionar o fluxo de dados pela associação de um outro ficheiro ou dispositivo físico a um ficheiro lógico

Ficheiros de texto

Um ficheiro de texto

- É um ficheiro de carateres
- É um *ficheiro binário* em que cada caráter ocupa 1 byte
- Tem a particularidade de ser **legível** pelos seres humanos, quando abertos por um processador de texto (por exemplo, pico, notepad, …)

Funções para manipulação de ficheiros

Abertura de um ficheiro

fopen

Fecho de um ficheiro

fclose

Escrita formatada num ficheiro

fprintf

Leitura formatada de um ficheiro

fscanf

Escrita de carateres num ficheiro

fputc

Leitura de carateres de um ficheiro

fgetc

Testar o indicador de final de ficheiro

feof

Biblioteca

stdio.h

Abertura de um ficheiro

Sintaxe

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

- Parâmetros da função
  - **filename**: string que contém o nome do ficheiro físico (em disco)
  - **mode**: string que contém o modo de abertura do ficheiro, que pode ser
    - **"r"** para leitura (read)
    - **"w"** para escrita (write)
    - **"a"** para escrita no fim do ficheiro (acrescentar)
- Devolução/retorno da função
  - **endereço** de um FILE, se não houve problemas na abertura
  - **NULL**, em caso de erro na abertura

Sintaxe

- Modos de abertura de um ficheiro
  - **"r"** (read) → abertura de ficheiro para leitura
    - se o ficheiro existe, coloca o marcador no início
    - se o ficheiro não existe, devolve um erro de abertura
  - **"w"** (write) → abertura de ficheiro para escrita
    - se o ficheiro existe, esvazia-o e coloca o marcador no início
    - se o ficheiro não existe, então cria-o e coloca o marcador no início
  - **"a"** (append) → abertura de ficheiro para escrita no fim do ficheiro (acrescentar)
    - se o ficheiro existe, coloca o marcador no fim do ficheiro
    - se o ficheiro não existe, então cria-o e coloca o marcador no início (= fim)
- É possível também abrir um ficheiro para leitura e escrita em simultâneo
  - basta acrescentar o sinal "+" ao modo de abertura do ficheiro ("r+")

Exemplo

```
#include <stdio.h>
void main()
{
  FILE  *f;
  f = fopen("teste.txt", "r");
  if (f == NULL)
      printf("Erro na abertura do ficheiro!\n"):
  else
      printf("Abertura de ficheiro com sucesso!\n");
}
```

Fecho de um ficheiro

Sintaxe

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

- Parâmetros da função
  - **file**: variável que contém o endereço do ficheiro lógico
- Devolução da função (normalmente ignorado)
  - **0** (zero), se não houve problemas no fecho
  - **EOF** (caracter de fim de ficheiro - "End-Of-File"), em caso de erro no fecho

Exemplo

```
#include <stdio.h>
void main()
{
  FILE  *f;
  f = fopen("teste.txt", "r");
  if (f == NULL)
     printf("Erro na abertura do ficheiro!\n");
  else
  {
     printf("Abertura de ficheiro com sucesso!\n");
     fclose(f);
  }
}
```

Escrita formatada num ficheiro

Sintaxe

int fprintf (FILE ***file**, const char ***formato**, tipo **var_1**, …, tipo **var_k**);

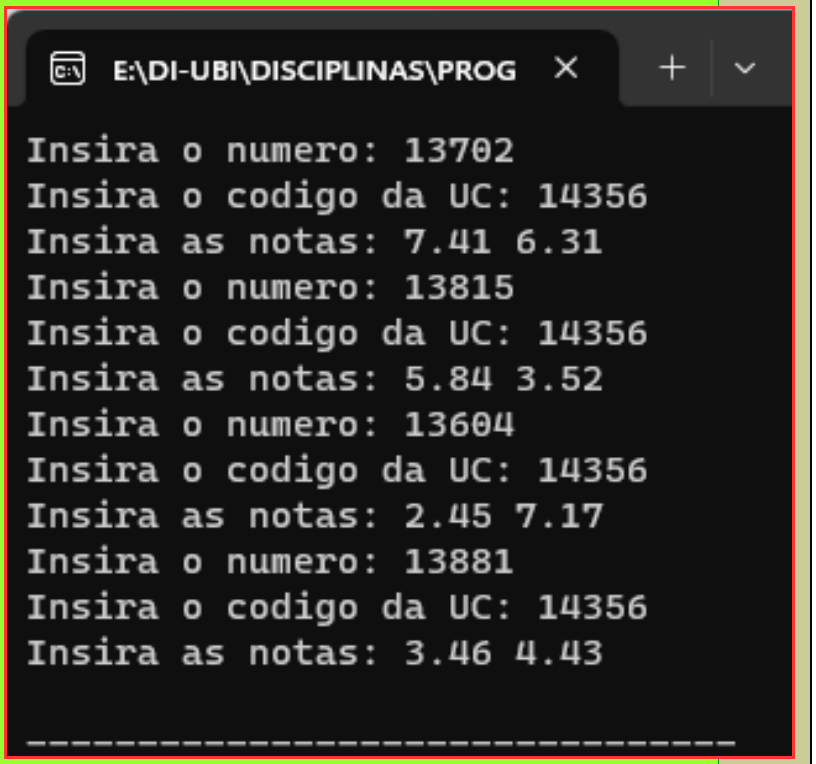
- Parâmetros da função
  - **file**: variável que contém o endereço do ficheiro lógico
  - **formato**: string com os k formatos de tipos de dados a escrever no ficheiro
  - **var_1**, …, **var_k**: as k variáveis que contêm os k dados a escrever
- Devolução da função (normalmente ignorado)
  - o **número** de dados escritos corretamente (inteiro menor ou igual a k)
  - **EOF** (caracter de fim de ficheiro - "End-Of-File"), em caso de erro de escrita

Exemplo

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int  Numero;
   int  CodigoUC;
   float  Notas[2];
} ESTUDANTE;
...
```

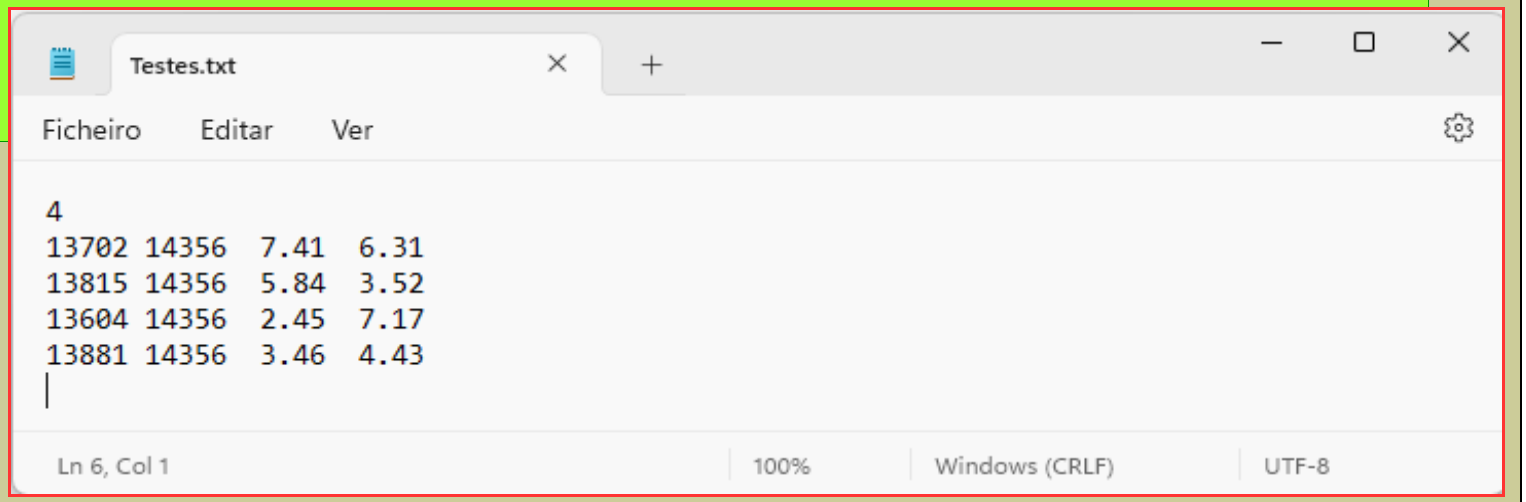
Exemplo

```
...
void main()
{
  int  k, TAM = 4;
  ESTUDANTE  *PROG;
  FILE  *fout;
  PROG = malloc(TAM * sizeof(ESTUDANTE));
  for(k = 0; k < TAM; k++)
  {
     printf("Insira o numero: ");
     scanf("%d", &PROG[k].Numero);
     printf("Insira o codigo da UC: ");
     scanf("%d", &PROG[k].CodigoUC);
     printf("Insira as notas: ");
     scanf("%f%f", &PROG[k].Notas[0], &PROG[k].Notas[1]);
  }
  ...
```



Exemplo

```
...
fout = fopen("Testes.txt", "w");
fprintf(fout, "%d\n", TAM);
for (k = 0; k < TAM; k++)
{
    fprintf(fout, "%d ", PROG[k].Numero);
    fprintf(fout, "%d ", PROG[k].CodigoUC);
    fprintf(fout, "%5.2f %5.2f\n", PROG[k].Notas[0], PROG[k].Notas[1]);
}
fclose(fout);
}
```



Leitura formatada de um ficheiro

Sintaxe

int fscanf (FILE ***file**, const char ***formato**, tipo ***var_1**, …, tipo ***var_k**);

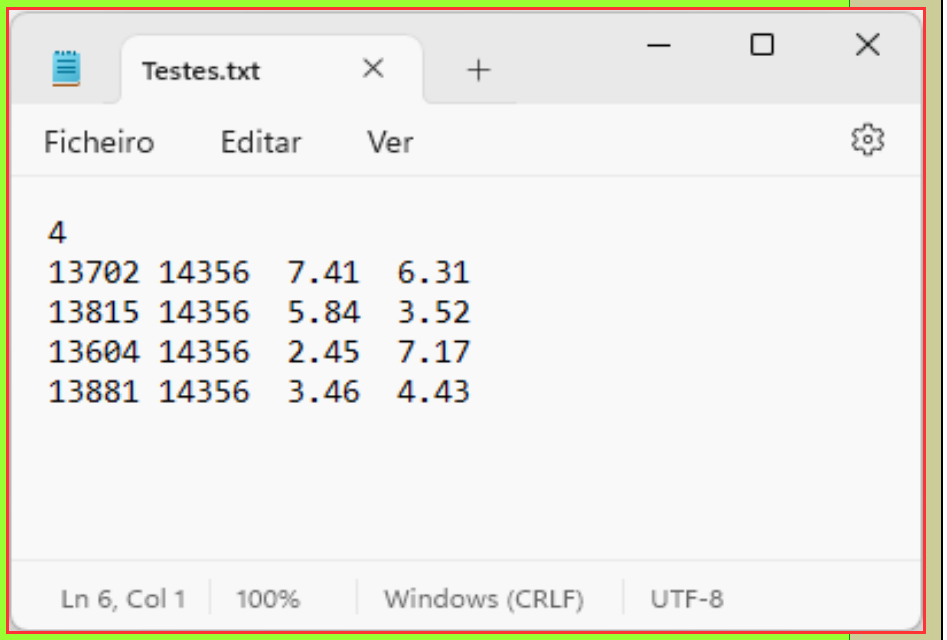
- Parâmetros da função
  - **file**: variável que contém o endereço do ficheiro lógico
  - **formato**: string com os k formatos de tipos de dados a ler do ficheiro
  - **var_1**, …, **var_k**: endereços das k variáveis que recebem os k dados a ler
- Devolução da função
  - o **número** de dados lidos corretamente (inteiro menor ou igual a k)
  - **EOF** (caracter de fim de ficheiro - "End-Of-File"), em caso de erro de leitura

Exemplo

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int  Numero;
   int  CodigoUC;
   float  Notas[2];
} ESTUDANTE;
...
```

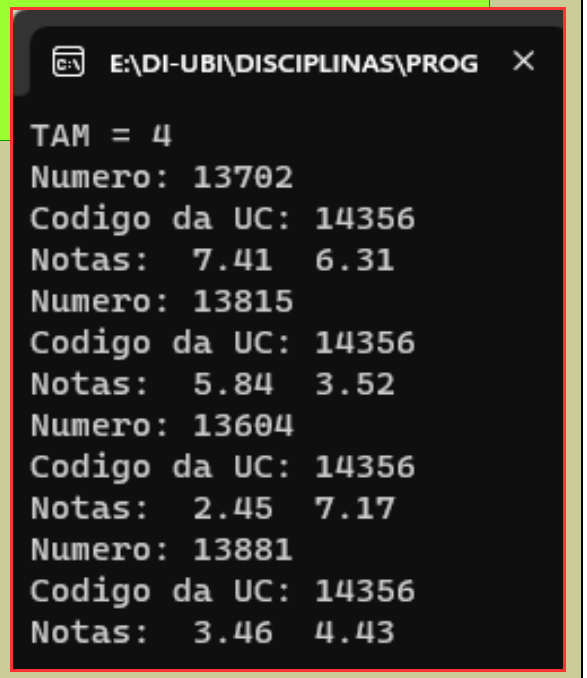
Exemplo

```
...
void main()
{
  int  k, TAM;
  ESTUDANTE  *PROG;
  FILE  *fin;
  fin = fopen("Testes.txt", "r");
  fscanf(fin, "%d", &TAM);
  PROG = malloc(TAM * sizeof(ESTUDANTE));
  for(k = 0; k < TAM; k++)
  {
     fscanf(fin, "%d", &PROG[k].Numero);
     fscanf(fin, "%d", &PROG[k].CodigoUC);
     fscanf(fin, "%f%f", &PROG[k].Notas[0], &PROG[k].Notas[1]);
  }
  fclose(fout);
  ...
```



Exemplo

```
  ...
  printf("TAM = %d\n", TAM);
  for (k = 0; k < TAM; k++)
  {
     printf("Numero: %d\n", PROG[k].Numero);
     printf("Codigo da UC: %d\n", PROG[k].CodigoUC);
     printf("Notas: %5.2f %5.2f\n", PROG[k].Notas[0], PROG[k].Notas[1]);
  }
}
```



Escrita de carateres num ficheiro

Sintaxe

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

- Parâmetros da função
  - **ch**: variável que contém o caráter a escrever no ficheiro
  - **file**: variável que contém o endereço do ficheiro lógico
- Devolução da função (normalmente ignorado)
  - o **caráter** contido na variável **ch**, se a operação de escrita teve sucesso
  - **EOF** (caracter de fim de ficheiro - "End-Of-File"), em caso de erro na escrita

Leitura de carateres de um ficheiro

Sintaxe

```
int  fgetc (FILE *file);
```

- Parâmetros da função
  - **file**: variável que contém o endereço do ficheiro lógico
- Devolução da função
  - o **caráter** lido em forma de inteiro
  - **EOF** (caracter de fim de ficheiro - "End-Of-File"), em caso de ficheiro vazio

Teste de indicador de final de ficheiro

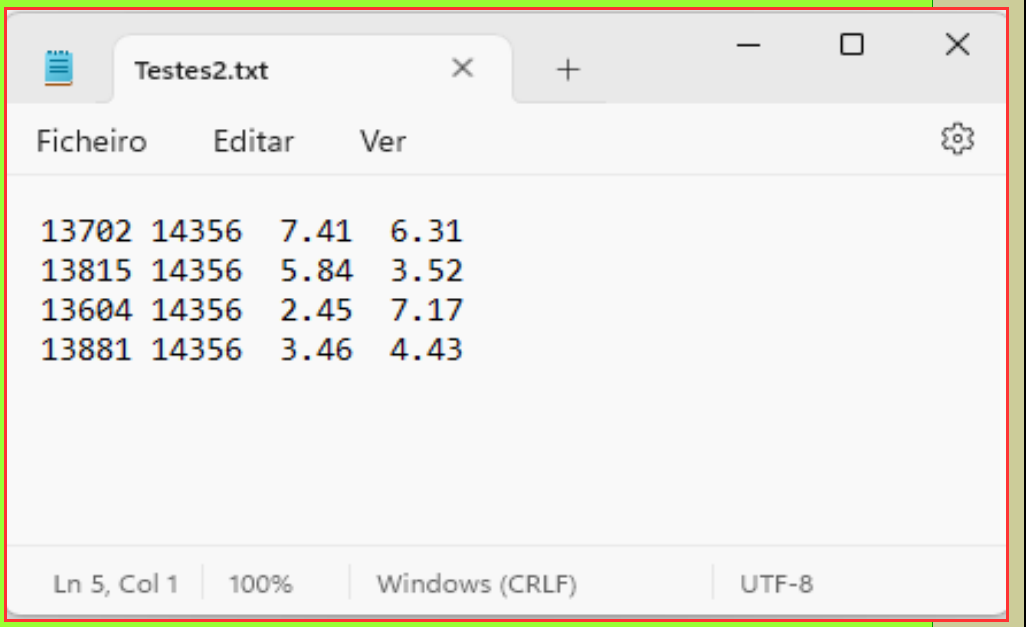
Sintaxe

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

- Parâmetros da função
  - **file**: variável que contém o endereço do ficheiro lógico
- Devolução da função
  - **0**, se o indicador de final de ficheiro não foi atingido
  - **valor inteiro** diferente de 0, se o indicador de final de ficheiro foi atingido

Exemplo

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int  Numero;
   int  CodigoUC;
   float  Notas[2];
} ESTUDANTE;
void main()
{
   int  k, TAM = 0;
   ESTUDANTE  *PROG;
   char c;
   FILE  *fin;
   fin = fopen("Testes2.txt", "r");
   ...
```



Exemplo

```
...
PROG = malloc(TAM * sizeof(ESTUDANTE));
c = fgetc(fin);
while (feof(fin) == 0)  // ou while (!feof(fin)) ou while (c != EOF)
{
   TAM = TAM + 1;
   PROG = realloc(PROG, TAM * sizeof(ESTUDANTE));
   fscanf(fin, "%d", &PROG[TAM-1].Numero);
   fscanf(fin, "%d", &PROG[TAM-1].CodigoUC);
   fscanf(fin, "%f%f", &PROG[TAM-1].Notas[0], &PROG[TAM-1].Notas[1]);
   c = fgetc(fin);    // ler o '\n'
   c = fgetc(fin);    // ler o carater que vem a seguir ao '\n'
}
fclose(fout);
...
```

Exemplo

```
  ...
  printf("TAM = %d\n", TAM);
  for (k = 0; k < TAM; k++)
  {
     printf("Numero: %d\n", PROG[k].Numero);
     printf("Codigo da UC: %d\n", PROG[k].CodigoUC);
     printf("Notas: %5.2f %5.2f\n", PROG[k].Notas[0], PROG[k].Notas[1]);
  }
}
```