Módulo 3

- Estruturas
- Ficheiros

Structs

Structs

Campos ou Membros de uma Estrutura

As componentes armazenadas dentro de uma estrutura são denominadas por **campos** ou **membros** da estrutura.

Motivação

Imaginemos que desejamos tratar dados relativos a uma pessoa. Poderíamos criar um conjunto de variáveis com a informação (registo) da pessoa:

- `char nome[100];`
- `char sexo, estado_civil;`
- `unsigned int idade;`
- `float salario;`

Estruturas: permitem que toda esta informação esteja agrupada e relacionada entre si.

Structs - sintaxe

Sintaxe da declaração de uma estrutura

```
struct [nome_da_estrutura]
{
    tipo_1 campo_1a, campo_1b;
    tipo_2 campo_2;
    ...
    tipo_n campo_n;
};
```

A declaração de uma estrutura define um tipo de dados composto, mas não define variáveis de tipo estrutura.

Exemplo: uma estrutura capaz de armazenar uma data

```
struct Data
{
    int dia, ano; // campos numéricos
    char mes[12]; // uma string para o mês
};
```

Structs - Inicialização - declaração

Sintaxe da declaração com inicialização

```
struct nome_da_estrutura nome_var =
{valor_1, valor_2, ..., valor_n};
```

onde se assume que `n` é o número de membros da estrutura em causa, e os valores dados correspondem aos membros da *struct* pela ordem em que estes surgem na declaração da mesma.

Considere-se a **declaração**: `struct Data dt, datas[5], *ptr_data;`

- **dt** é uma variável do tipo `struct Data`, contendo 2 inteiros e uma string com espaço para 11 char úteis.
- **datas** é uma tabela de 5 elementos, cada um deles contendo uma estrutura do tipo `struct Data` ou seja, é reservado espaço para 5 elementos do tipo `struct Data`.
- **ptr_data** é um ponteiro para o tipo `struct Data`.

```
struct Data
{
  int dia, ano;
  char mes[12];
};
```

Structs - exemplo

```c
#include <stdio.h>
int main() {

struct Data {
  int dia, ano;
  char mes[12];
} dx = {10, 2099, "Outubro"}, tab[2];

  struct Data dy = {25, 2035, "Abril"};
  tab[0] = dx; tab[1] = dy;
  // Soma 1 ao ano de tab[0]
  tab[0].ano++;
  // Mostra tab
  printf("A tabela de datas:\n");

  for(int i = 0; i < 2; i++)
        printf(" %d / %s / %d  \n", tab[i].dia, tab[i].mes, tab[i].ano);

  printf("A primeira letra de cada mes:\n");

  for(int i = 0; i < 2; i++)
        printf(" %c eh a 1a letra de %s  \n", tab[i].mes[0], tab[i].mes);

return (1);}
```

```
A tabela de datas:
 10 / Outubro / 2100
 25 / Abril / 2035
A primeira letra de cada mes:
 O eh a 1a letra de Outubro
 A eh a 1a letra de Abril
```

Operadores e estruturas

- Sendo uma estrutura definida pelo utilizador, não podemos utilizar os operadores relacionais

  `<, >, >=, <=, ==` ou `! =`

- Da mesma forma não podemos usar operadores aritméticos.

- Mas podemos usar o operador atribuição (`=`), **para copiar uma estrutura** como um bloco de dados para outra estrutura do mesmo tipo (**não precisamos copiar membro a membro**).

- Podemos obter o endereço de uma estrutura, ou o endereço de um membro de uma estrutura p/ex:

  - Se `v` é uma estrutura então `&v` devolve o endereço dessa estrutura em memória (ou seja o menor dos endereços que ela ocupa)

  - Se `x` é um membro de `v`, `&v.x` devolve o endereço de memória de `v.x`

// Ficheiros

Ficheiros – *fopen*

A função *fopen* realiza a **abertura de um ficheiro**.

Protótipo da função *fopen*

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

- `filename`: string contendo o nome físico do ficheiro a abrir.
- `mode`: string contendo o modo de abertura do ficheiro.

`fopen`: retorna ponteiro para estrutura FILE em caso de sucesso, e retorna NULL caso contrário.

O `filename` (nome do ficheiro) pode ser o nome completo, ou local (ou relativo) ao ponto de execução do ficheiro, e depende do sistema operativo.

Exemplos:

C:\ProgComp\dados.txt

"/home/username/dados.txt"

Ficheiros – *fopen* (mode: modos de abertura)

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

Há **três modos básicos** de abertura de um ficheiro:

- `"r"`: read – abertura de ficheiro para **leitura**.

Caso não seja possível abrir o ficheiro, a função de abertura (*fopen*) devolve o valor NULL.

- `"w"`: write – abertura de ficheiro para **escrita**.

É criado um novo ficheiro para escrita. Se este já existia é **apagado e reescrito**.
Caso não seja possível criar o ficheiro, a função *fopen* devolve o valor NULL.

- `"a"`: append – abertura de ficheiro para **escrita** em modo de acrescento.

Se o ficheiro existe, posiciona-se no final do ficheiro para que a escrita seja feita partir desse ponto.
Caso não seja possível criar o ficheiro, o *fopen* devolve o valor NULL

Ficheiros – modos de abertura (escrita e leitura)

Operações de **escrita e leitura** em simultâneo , colocando um **+** a seguir ao modo:

"`r+`": *read* – abertura de ficheiro para **leitura e escrita**.
Se o ficheiro não existir ocorre um erro. Se este já existia os novos dados serão escritos a partir do início do ficheiro, substituindo a informação anterior (reescrita parcial ou total).

"`w+`": *write* – abertura de ficheiro para **leitura e escrita**. É criado um novo ficheiro. Se este já existia é apagado e reescrito.

"`a+`": *append* – abertura de ficheiro para **leitura e escrita**. Se o ficheiro existe, posiciona-se no final do ficheiro para que a escrita seja feita partir desse ponto.

Modo (`mode`)	Tipo	Leitura ?	Escrita ?	Se ficheiro não existir…	Se ficheiro existe…	Posição inicial
`r`	Leitura	Sim	**Não**	NULL	OK	Início
`w`	Escrita	Não	Sim	**Cria**	**Recria**	Início
`a`	Acrescento	Não	Sim	**Cria**	OK	Fim
`r+`	Leitura-Escrita	Sim	Sim	NULL	**Altera**	Início
`w+`	Leitura-Escrita	Sim	Sim	**Cria**	**Recria**	Início
`a+`	Leitura-Escrita	Sim	Sim	**Cria**	Acrescenta	Fim

Ficheiros – escrita de caracteres: *fputc*

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

- Escreve o char caractere `ch` no stream `fileptr` (que se assume já foi previamente associado a um ficheiro através da função *fopen*).
- Em caso de sucesso devolve o caractere `ch`; e **EOF** se houver erro.

Ficheiros – leitura de caracteres: *fgetc*

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

- Retorna o caractere obtido se a leitura foi realizada com sucesso ou `EOF` se houver erro
- Se o erro for devido a atingir o **fim de ficheiro**, é ativada a *flag* de fim de ficheiro (cf. função *feof*), para esse *stream*. Se a causa for **outra, activa o indicador de erro** (cf. função *ferror*), para esse *stream*.
- Note que a função devolve um `int` e não um *char* para ter a possibilidade de devolver, além de um qualquer *char*, a constante `EOF`.
- Sempre que é pedida a leitura de um char é devolvido o byte corrente se existir ou EOF em caso contrário; em seguida o cursor de leitura tenta avançar para o byte seguinte.

Ficheiros - *fclose*

O fecho do ficheiro consiste em anular a relação entre a variável do nosso programa e o ficheiro.

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

filepointer: ponteiro para a variável FILE associada ao ficheiro por uma invocação anterior da função fopen.

Retorna 0 em caso de sucesso ou a constante EOF em caso de erro.

Antes do ficheiro ser fechado são gravados, fisicamente, no ficheiro todos os dados que ainda pudessem estar em bufffers associados ao ficheiro. Adicionalmente, é libertada a memória reservada pela *fopen*.

Ficheiros – exemplo de leitura em modo TEXTO

```c
#include <stdio.h>
int main() {
  FILE *fp = NULL;
  char ch='\0', my_file[50]="features.txt"; //tipo TXT

  printf("Vamos ler o conteudo do ficheiro:");
  printf(my_file);
  putchar('\n');

  fp = fopen(my_file,"r"); // LER ficheiro "r"
  if (fp==NULL) {
        printf("Impossível Abrir o ficheiro %s \n",my_file);
        return(-1);
  }

//Imprimir conteudo
  while (ch != EOF){
    ch = fgetc(fp);
    putchar(ch);//imprime 1 caractere
  }

  fclose(fp); //fechar o FILE *

  return (1);
}
```

```
1 48.538208 9.440077  1.102703  0.021723  0.059699  0.456924
0 57.936264 12.603884  1.997085  0.051343  0.149340  0.046218
0 48.105747 13.858523  1.091288  0.005394  0.479100  0.483645
0 24.271097 8.507536  0.466721  0.278121  0.182137  0.161543
0 21.337679 12.698994  0.442697  25.332808  1.285921  0.179461
```

```
C:\WINDOWS\SYSTEM32\cmc
Vamos ler o conteudo do ficheiro:features.txt
1 48.538208 9.440077  1.102703  0.021723  0.059699  0.456924
0 57.936264 12.603884  1.997085  0.051343  0.149340  0.046218
0 48.105747 13.858523  1.091288  0.005394  0.479100  0.483645
0 24.271097 8.507536  0.466721  0.278121  0.182137  0.161543
0 21.337679 12.698994  0.442697  25.332808  1.285921  0.179461
```

Ficheiros – leitura e escrita formatadas

Podemos ler escrever em ficheiros de forma semelhante à leitura do teclado e escrita no ecrã:

- Usando as funções **`fscanf`** e **`fprintf`**

```
int fscanf(FILE * fileptr, const char *format, ...);
int fprintf(FILE * fileptr, const char *format, ...);
```

- *fscanf* devolve `EOF` se detetar o fim de ficheiro ou o número de parâmetros que conseguiu ler.
- *fprintf* devolve o número de *char* enviados para o stream de saída, ou um código de erro se ocorreu algum problema de conversão.

Nota: A função `fscanf` trata os *White Space* da mesma forma que a função *scanf*.

Exemplo - *fprintf*

```
#include <stdio.h>
int main() {
  FILE *fp = NULL;
  char my_file[50]="Ex.txt"; //tipo TXT
  float x, y;
  int i=1;
  printf("Utilizacao do fprintf:");
  fp = fopen(my_file,"w"); //cria ficheiro-escrita "w"
  if (fp==NULL)  {
        printf("Impossível Criar o ficheiro %s \n",my_file);
        return(-1); }

  printf(" Digite dois numeros reais: \n " ) ;
  scanf ("%f %f", &x, &y);

//Escrita formatada no ficheiro
  fprintf(fp," (%d) Soma: %1.2f + %1.2f = %1.4f \n", i, x, y, x+y);
  i++;
  fprintf(fp," (%d) Mult.: %1.2f * %1.2f = %1.4f \n", i, x, y, x*y);
  i++;
  fprintf(fp," (%d) Div: %1.2f / %1.2f = %1.4f \n", i, x, y, x/y);

  fclose(fp); //Fecha o FILE *
  return (1);
}
```

```
Utilizacao do fprintf:
 Digite dois numeros reais:
5.55
3.33
```

Ex - Bloco de notas

Ficheiro Editar Ver

```
(1) Soma: 5.55 + 3.33 = 8.8800
(2) Mult.: 5.55 * 3.33 = 18.4815
(3) Div: 5.55 / 3.33 = 1.6667
```

Exemplo - *fscanf*

```c
#include <stdio.h>
int main() {
  FILE *fp = NULL;
  char my_file[50]="Ex.txt"; //tipo TXT
  char Str[100]="";        int i=1;

  printf("Utilizacao do fscanf:");
  fp = fopen(my_file,"r"); //abre modo leitura "r"
  if (fp==NULL)  {
    printf("Impossível Abrir o ficheiro %s \n",my_file);
    return(-1); }

  //Leitura do ficheiro com fscanf
  while(fscanf(fp, "%s", Str) != EOF)
  {
    printf("String %i: %s \n", i, Str);
    i++;
  }

  fclose(fp); //Fecha o FILE *

  return (1);
}
```

Atenção aos *White Spaces !*

```
(1) Soma: 5.55 + 3.33 = 8.8800
(2) Mult.: 5.55 * 3.33 = 18.4815
(3) Div: 5.55 / 3.33 = 1.6667
```

C:\WINDOWS\SYSTEM32\cmc

```
Utilizacao do fscanf:
String 1: (1)
String 2: Soma:
String 3: 5.55
String 4: +
String 5: 3.33
String 6: =
String 7: 8.8800
String 8: (2)
String 9: Mult.:
String 10: 5.55
String 11: *
String 12: 3.33
String 13: =
String 14: 18.4815
String 15: (3)
String 16: Div:
String 17: 5.55
String 18: /
String 19: 3.33
String 20: =
String 21: 1.6667
```

Exemplo – *fprintf*, *fscanf*


```
#include <stdio.h>
int main() {
FILE *fp = NULL;
char my_file[50]="ficheiroExe4.txt"; //tipo TXT
char Str[100]="Numero PI= 3.1415926";
char ReadStr1[10]; char ReadStr2[10];
float X; char ch; int k;

fp = fopen(my_file,"w"); //abre ficheiro em modo escrita "w"
if (fp==NULL)  {
          printf("Impossível Criar/Abrir o ficheiro %s \n",my_file);
          return(-1); }
fprintf(fp,"%s", Str);
fclose(fp); //Fecha o FILE *

puts("Utilizacao do fscanf:");
fp = fopen(my_file,"r"); //abre ficheiro em modo leitura "r"
if (fp==NULL)  {
          printf("Impossível Abrir o ficheiro %s \n",my_file);
          return(-1); }
//Leitura formatada com fscanf
fscanf (fp, "%s", ReadStr1);
fscanf (fp, "%s", ReadStr2);
fscanf (fp, "%d", &k);
fscanf (fp, "%c", &ch);
fscanf (fp, "%f", &X);
printf(" %s %s %d %c %f \n",ReadStr1,ReadStr2, k,ch,X);

fclose(fp); //Fecha o FILE *
return (1); }
```

Atenção aos *White Spaces !*

Ficheiros – leitura e escrita formatadas: *fputs*

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

- permite enviar para um ficheiro de texto uma linha de cada vez.
- Escreve todos os caracteres da string `str` para o fluxo de saída.
- O caractere de terminação de `str` não é gravado.

- `str` - string (devidamente terminada) a escrever no ficheiro.
- `stream` - fluxo de saída.

- Em caso de sucesso devolve um número não negativo. Em caso de erro devolve EOF e activa *flag* de erro (ver função *ferror*).

Ficheiros – leitura e escrita formatadas: *fgets*

```
char *fgets( char *str, int count, FILE *stream );
```

- permite ler um ficheiro de texto, lendo uma linha de cada vez.

- Lê no máximo count-1 char do fluxo de entrada stream para a str, ou até encontrar `\n`, ou até EOF.
- O caractere de fim de linha (`\n`) é armazenado em str, se foi este que determinou o fim de leitura da linha.
- O caractere `\0` é escrito após o último caracter lido.
- Em caso de sucesso devolve o ponteiro str, e o ponteiro NULL em caso de erro.

- Se o erro tiver sido causado pela condição de fim de arquivo, ativa a flag de fim de ficheiro (consulte ver feof) no fluxo. O conteúdo da tabela apontado por str não é alterado neste caso.
- Se o erro tiver sido causado por algum outro erro, define o indicador de erro (ver ferror) no fluxo. O conteúdo da tabela apontada por str é indeterminado (pode até não ser terminado pelo caractere `'\0'`).

Exemplo – *fputs* *fgets*

```
. . .
FILE *fp = NULL;
char S1[100]="";

fp = fopen("FicheiroExe6.txt","w");
if (fp==NULL)  {
          printf("Impossível Criar/Abrir o ficheiro! \n");
          return(-1); }

puts("Digite uma frase:");
fgets(S1, sizeof(S1), stdin);
fputs(S1,fp);
fputc('\n',fp); //fputs nao adiciona \n
puts("Digite outra frase:");
fgets(S1, sizeof(S1), stdin);
fputs(S1,fp);
fclose(fp); //Fecha o FILE *
//================================
puts(" ****** exemplo com fgets ****** ");
fp = fopen("FicheiroExe6.txt","r");
if (fp==NULL)  {
          printf("Impossível Abrir o ficheiro! \n");
          return(-1); }
printf("%s",fgets(S1,7,fp));
puts("---> segundo fgets:");
printf("%s",fgets(S1,50,fp));
fclose(fp); //Fecha o FILE *

return (1); }
```

Testar sem o
`fputc('\n',fp)`