

# Programação 2024/25

## LEI, LEI-PL, LEI-CE

### Aula Laboratorial 5

#### Bibliografia:

K. N. King. *C programming: A Modern Approach* (2<sup>nd</sup> Edition). W. W. Norton: capítulos 15 e 16.

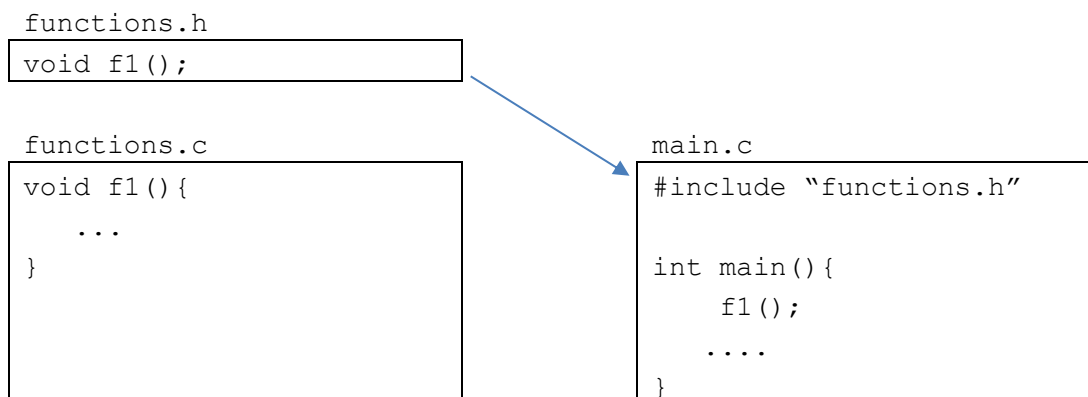
#### Código de apoio para a aula:

<https://gist.github.com/FranciscoBPereira>

#### Ficheiros de Cabeçalho (Header Files)

É possível criar um programa em C a partir de código fonte distribuído por vários ficheiros. Esta divisão tem várias vantagens, nomeadamente na organização, eficiência e reutilização de código. Existem regras que devem ser seguidas para criar um programa único a partir de vários ficheiros com código fonte.

1. O código pode estar dividido por vários ficheiros contendo código fonte. Estes ficheiros têm extensão `.c`. A função `main()` estará num destes ficheiros.
2. Os ficheiros de cabeçalho permitem estabelecer a comunicação entre os diferentes módulos que contêm o código fonte. Estes ficheiros têm extensão `.h` e estão normalmente associados a ficheiros `.c`, publicitando o que é disponibilizado ao programa por eles. Podem conter:
  - a. Protótipos de funções;
  - b. Definição de tipos;
  - c. Declaração externa de variáveis globais.
3. A diretiva `#include` sinaliza que um ficheiro `.c`, durante o processo de compilação, deve obter informação relevante que existe num outro ficheiro `.c`. Ocorre, por exemplo, quando uma função é chamada num módulo `.c`, tendo sido escrita num outro ficheiro.



## Programação 2024/25

### LEI, LEI-PL, LEI-CE

4. A diretiva `#include` pode ter dois formatos: `#include <abcd.h>` ou `#include "abcd.h"`. A diferença indica ao compilador onde deve ser localizado o ficheiro de cabeçalho.
5. O aumento da dimensão de um programa, com vários ficheiros de código e de cabeçalho, pode criar situações de inclusão múltipla. Para evitar este erro, os ficheiros de cabeçalho devem ser protegidos com o par de diretivas `#ifndef ... #endif`.

functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

...

#endif
```

**Nota:** é essencial consultar o capítulo 15 do livro “*C Programming: A Modern Approach*” para obter informações mais detalhadas sobre este tópico.

### Estruturas

#### Exercícios Obrigatórios

1. Pretende-se representar um ponto num espaço cartesiano com 2 dimensões. O código deste exercício está distribuído por 3 ficheiros:

- i. *main.c*: contém a função *main()*.
  - ii. *ponto.c*: contém todas as funções que manipulam as variáveis do tipo ponto.
  - iii. *ponto.h*: Permite efetuar a partilha de informação entre os 2 ficheiros com código fonte.
- a) Crie um tipo estruturado que lhe permita armazenar as coordenadas inteiras (x, y) de um ponto. A criação do tipo deve ser efetuada no ficheiro *ponto.h*.
  - b) Escreva uma função em C que imprima as coordenadas de um ponto recebido como parâmetro.
  - c) Escreva uma função em C que inicialize uma estrutura do tipo ponto com coordenadas indicadas pelo utilizador. A função recebe o endereço do ponto a inicializar.
  - d) Escreva uma função em C que desloque um ponto no espaço 2D. A função recebe, como parâmetros, o endereço do ponto a mover e o valor dos deslocamentos ao longo dos eixos *xx* e *yy*.

## Programação 2024/25

### LEI, LEI-PL, LEI-CE

- e) Escreva uma função em C que determine o quadrante do plano cartesiano a que pertence um ponto recebido como parâmetro.
- f) Escreva uma função em C que verifique se 3 pontos recebidos como parâmetros estão alinhados na mesma reta. A função devolve 1 se pertencerem à mesma reta, ou 0, caso contrário.

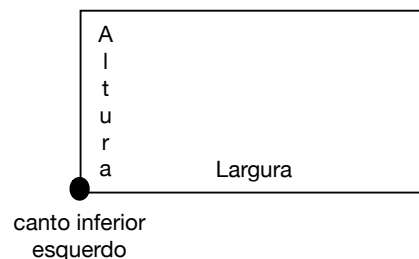
**Sugestão:** para responder a esta questão, calcule a equação reduzida da reta que passa por 2 dos pontos recebidos. Depois disso, verifique se o terceiro ponto pertence a essa reta.

- g) Escreva uma função em C que calcule a distância Euclidiana entre 2 pontos recebidos como parâmetros. A função devolve o valor calculado como resultado.

2. Pretende-se representar um retângulo num espaço com 2 dimensões. O retângulo deve ser definido através do seu canto inferior esquerdo, altura e largura. A altura e a largura são valores inteiros positivos. Ao responder a esta questão, deve utilizar a estrutura definida no exercício 1. O código deste exercício está distribuído por 2 ficheiros que devem ser adicionados ao projeto da pergunta anterior:

- i. *retangulo.c*: contém todas as funções que manipulam as variáveis do tipo retângulo.
- ii. *retangulo.h*: Permite efetuar a partilha de informação entre os vários ficheiros com código fonte.

- a) Crie um tipo estruturado que lhe permita representar um retângulo através do ponto que identifica o canto inferior esquerdo, a altura e a largura. A criação do tipo deve ser efetuada no ficheiro *retangulo.h*.



- b) Escreva uma função em C que imprima as coordenadas dos 4 cantos de um retângulo recebido como parâmetro.
- c) Escreva uma função em C que inicialize uma estrutura do tipo retângulo com informação indicada pelo utilizador. A função recebe o endereço do retângulo a inicializar.


## Programação 2024/25

### LEI, LEI-PL, LEI-CE

- d) Escreva uma função em C que devolva a área de um retângulo passado como parâmetro.
- e) Escreva uma função em C que verifique se um ponto está dentro de um retângulo (excluindo as arestas que definem a sua fronteira). A função recebe como parâmetros o retângulo e o ponto. Devolve 1 se o ponto estiver dentro do retângulo, ou 0, caso contrário.
- f) Escreva uma função em C que verifique se 2 retângulos recebidos como parâmetro se interseçam. A função devolve 1 se existir interseção, ou 0, caso contrário.

#### Exercícios Complementares

- g) Escreva uma função em C que desloque um retângulo no plano. A função recebe três parâmetros: o endereço do retângulo a deslocar, o deslocamento ao longo do eixo *xx* e o deslocamento ao longo do eixo *yy*.
- h) Escreva uma função em C que calcule o valor IoU (*Intersection Over Union*) para 2 retângulos recebidos como parâmetro. O valor calculado é devolvido como resultado.


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

# Programação 2024/25

## LEI, LEI-PL, LEI-CE

### Trabalho Autónomo – Unions

Uma *union* em C permite criar tipos estruturados, em que todos os campos estão sobrepostos. Desta forma apenas um campo pode ser utilizado em cada momento. Embora pareça um conceito estranho têm utilidade em várias situações, nomeadamente na flexibilização da organização de tipos estruturados. Devem consultar o livro *C Programming: A Modern Approach*, secção 16.4, para uma melhor compreensão do que é uma *union* e de como pode ser usada. Depois disso devem completar o exercício descrito a seguir.

3. Crie um pequeno programa que efetue a gestão completa de um vetor de números, sendo que este vetor em algumas posições guarda inteiros, enquanto noutras guarda reais do tipo *float*.

O tipo estruturado descrito a seguir permite criar um vetor com essas características. Tem 2 campos, um inteiro e um real, sendo que apenas um deles é usado em cada posição do vetor. A estrutura contém também uma flag inteira chamada *tipo* que indica se uma determinada variável deste tipo está a guardar um inteiro (valor 0) ou um real (valor 1).

```
typedef struct info numero;
struct info{
    int tipo;      //0:inteiro; 1:real
    union{
        int i;
        float x;
    } valor;
};
```

O programa gere um vetor com 1º elementos do tipo *numero*. Complete o código adicionando as 3 funções em falta:

- A primeira função preenche o vetor com valores especificados pelo utilizador. O valor indicado pode ser inteiro ou real e a função deve fazer o armazenamento de forma correta. Há duas maneiras para se descobrir se o valor que vai ser introduzido é inteiro ou real: a mais simples é perguntar ao utilizador que tipo de valor é que vai ser introduzido; em alternativa, o programa pode descobrir se o valor que acabou de receber é inteiro ou real.
- A segunda função mostra na consola os valores que foram previamente armazenados no vetor.
- A terceira função calcula as médias dos valores reais e dos valores inteiros armazenados no vetor.