Guião da Ficha Trabalho 2

Funções em C

As funções permitem dividir o código de um programa em blocos mais pequenos. Estes blocos são mais fáceis de escrever, de corrigir e podem eventualmente ser aproveitados noutros programas.

Mesmo no mais simples programa em C, já tivemos contacto com funções. Por exemplo, já usámos a função `printf`, que é uma função standard do C. A linguagem C traz um conjunto alargado de funções, como o `printf`, `scanf`, `fopen`, `strcat`, `malloc`, etc. Para usar estas funções, é necessário incluir o header onde consta a declaração da função. Para usar o `printf`, tivemos que incluir o header `stdio.h`, para dar a conhecer ao compilador o tipo retornado pela função `printf`.

Para além das funções standards do C, interessa-nos escrever as nossas próprias funções para organizar os programas.

Definir uma função

Primeiro exemplo: a função pi()

Vamos começar por definir uma função muito simples, chamada `pi`, que calcula o valor de π (sabendo que arctan(1) = π/4).

```
float pi() {
    float res;
    res = 4*atan(1);
    return res;
}
```

Na definição desta função `pi`, identificam-se quatro partes distintas:

- Tipo retornado pela função (`float`, neste caso)
- Nome da função (`pi`, neste exemplo)
- Parâmetros da função (esta função não recebe nenhum parâmetro)
- Corpo da função (é o bloco de código entre { }).

Considere o seguinte programa, onde consta a declaração e definição da função e a sua utilização, na função `main`:

```
#include <stdio.h>
#include <math.h>

float pi() {
    float res;
    res = 4*atan(1);
    return res;
}

int main() {
    printf("O valor de pi é: %f\n", pi());
    return 0;
}
```

Nota: a função pi pode ser compactada numa linha apenas, ficando:

```
float pi() {
    return 4*atan(1);
}
```

Nota: em C, existe uma constante pré-definida em `math.h`, designada `M_PI` que representa o valor de π.

Segundo exemplo: calcular o Índice de Massa Corporal

O índice de massa corporal é um indicador usado pelos serviços de saúde para identificar casos de obesidade ou de magreza, quando atinge valores demasiado altos ou baixos. O cálculo é baseado na altura e no peso da pessoa, sendo dado por: peso / altura (em metros)^2

A declaração e definição da função `imc` e a sua utilização será algo como:

```
#include <stdio.h>

float imc(float p, float a) {
    float res;
    res = p / (a*a);
    return res;
}

int main() {
    float peso, altura, indice;
    printf("Indique o seu peso (em Kg): ");
    scanf("%f",&peso);
    printf("Indique a sua altura (em m): ");
    scanf("%f",&altura);
    indice = imc(peso, altura);
    printf("O valor do seu IMC é: %f\n", indice);
    return 0;
}
```

Compilação separada

As funções permitem estruturar o código, como se disse, em blocos mais pequenos, mas fáceis de desenvolver, corrigir e reaproveitar.
No entanto, quando os programas começam a crescer, a simples divisão por funções não chega. Para permitir o desenvolvimento de programas mais complexos e com diferentes equipas a trabalhar no mesmo programa, o C permite-nos organizar os programas de diferentes ficheiros. Cada um destes ficheiros tem, tipicamente, um conjunto de funções relacionadas. Por exemplo, as funções relacionadas com o tratamento dos clientes, outro com as funções relacionadas com o tratamento dos produtos, etc.

Vamos recuperar a função `imc` para mostrar como funciona a compilação separada. Começamos por dividir o programa em duas partes: a parte principal, onde está a função `main` e onde se leem os dados do utente, e a parte onde é feito o cálculo do indicador `imc`.

O resultado é o seguinte:

principal.c	indicadores.c

`#include <stdio.h>`
 	
`float imc(float p, float a);`
 	
`int main() {`
`    float peso, altura, indice;`
`    printf("Indique o seu peso (em Kg): ");`
`    scanf("%f",&peso);`
`    printf("Indique a sua altura (em m): ");`
`    scanf("%f",&altura);`
`    indice = imc(peso, altura);`
`    printf("O valor do seu IMC é: %f\n", indice);`
`    return 0;`
`}`
	`float imc(float p, float a) {`
	`    float res;`
	`    res = p / (a*a);`
	`    return res;`
	`}`

Temos então o programa dividido em dois ficheiros separados. Note que em `principal.c` tivemos que acrescentar a declaração da função `imc`, já que a mesma é utilizada mais à frente.

Estes ficheiros podem ser compilados separadamente. Para gerar um executável, podemos fazer o seguinte:

Compilar (com a opção `-c`, o `gcc` só compila, não gera o executável):

```
gcc -c principal.c
gcc -c indicadores.c
```

Com estes dois comandos, são gerados os dois ficheiros `principal.o` e `indicadores.o`, cada um apenas com o código objeto resultante do código C que continham.

Para gerar o executável, é preciso ligar estes códigos objeto com as livrarias standards do C, com o comando (a opção `-o` é para indicar que queremos um executável chamado `imc`):

```
gcc -o imc principal.o indicadores.o
```

Temos o executável guardado em `imc`.

Exemplo de execução:

```
jgr@zoe:~/CLionProjects/li-aula-2$ ./imc
Indique o seu peso (em Kg): 70
Indique a sua altura (em m): 1,70
O valor do seu IMC é: 24.221453
```

Nota: pode-se reescrever a compilação em apenas dois comandos:

```
gcc -c indicadores.c
gcc -o imc principal.c indicadores.o
```

Compilação separada: com headers

No exemplo anterior já fomos capazes de separar um programa em funções e em ficheiros distintos, que depois são compilados de forma a gerar o programa pretendido.

Nesta estruturação dos programas por diferentes ficheiros, em vez de se incluir a declaração da função `imc` (em `principal.c`), inclui-se um header com essa declaração. Consegue-se, desta forma, uma melhor separação entre as partes de um programa. A equipa que desenvolve o programa principal apenas tem que incluir `indicadores.h` que é da responsabilidade de quem está a construir `indicadores.c`.

A melhor solução será fazer a seguinte separação:

`principal.c`

```
#include <stdio.h>
#include “indicadores.h”

int main() {
    float peso, altura, indice;
    printf("Indique o seu peso (em Kg): ");
    scanf("%f",&peso);
    printf("Indique a sua altura (em m): ");
    scanf("%f",&altura);
    indice = imc(peso, altura);
    printf("O valor do seu IMC é: %f\n", indice);
    return 0;
}
```

`indicadores.h`

```
float imc(float p, float a);
```

`indicadores.c`

```
float imc(float p, float a) {
    float res;
    res = p / (a*a);
    return res;
}
```

A compilação é feita da mesma forma:

```
gcc -c principal.c
gcc -c indicadores.c
gcc -o imc principal.o indicadores.o
```

Exercício 1: utilização do cmake

Ao introduzirmos o conceito de compilação separada, tivemos a necessidade de escrever vários comandos para gerar o executável. Quando os programas começam a crescer, começa a ser moroso e problemático saber o que é preciso recompilar, com que ordem, etc. Para sistematizar esse processo, as duas ferramentas muito utilizadas são: o `make` e o `cmake`.

O Clion que instalou utiliza o `cmake` para gerar o executável. Crie um novo projeto no CLion. Altere o `main.c` para ter o conteúdo de `principal.c` (do exercício anterior) e acrescente os outros dois ficheiros: `indicadores.h` e `indicadores.c`.

O resultado será:

main.c

```
#include <stdio.h>
#include “indicadores.h”

int main() {
    float peso, altura, indice;
    printf("Indique o seu peso (em Kg): ");
    scanf("%f",&peso);
    printf("Indique a sua altura (em m): ");
    scanf("%f",&altura);
    indice = imc(peso, altura*100);
    printf("O valor do seu IMC é: %f\n", indice);
    return 0;
}
```

indicadores.h

```
float imc(float p, float a);
```

indicadores.c

```
float imc(float p, float a) {
    float res;
    res = p / (a*a);
    return res;
}
```

O Clion gerou automaticamente um CMakeLists.txt com o seguinte conteúdo (onde li-aula-2 é o nome do projeto):

```
cmake_minimum_required(VERSION 3.13)
project(li_aula_2 C)
set(CMAKE_C_STANDARD 99)
add_executable(li_aula_2 main.c)
```

No entanto, com esta configuração o cmake não consegue gerar um executável, pois nós acrescentámos mais dois ficheiros ao projeto.

Com base na documentação do cmake em https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html, escreva um CMakeLists.txt adequado.

Scope nas funções (cuidados com strings/arrays)

Tenha em atenção que o âmbito das variáveis declaradas numa função, principalmente quando lida com string e genericamente com arrays. As variáveis que são declaradas numa função são criadas quando a função é invocada e são destruídas quando a função termina.

Considere a seguinte função, que devolve o signo dado o dia e o mês de nascimento:

```
char *signo(int dia, int mes) {
    char buffer[25];
    switch (mes) {
        case 1:
            strcpy(buffer, dia < 20 ? "Capricórnio" : "Aquário" );
            break;
        case 7:
            strcpy(buffer, dia < 23 ? "Cancer" : "Leao" );
            break;
        case 8:
            strcpy(buffer, dia < 23 ? "Leao" : "Virgem");
            break;
        case 9:
            strcpy(buffer, dia < 23 ? "Virgem" : "Libra");
            break;
        case 12:
            strcpy(buffer, dia < 22 ? "Sagitário" : "Capricórnio");
            break;
        default:
```

```
            strcpy(buffer, "desconhecido");
            break;
    }
    return buffer;
}
```

À primeira vista, está tudo bem. Mas não! **Esta função está ERRADA**. Assim que a função termina, a variável local `buffer` é destruída e o apontador que é retornado deixa de apontar para uma posição válida na memória.

Exercício 2: correção da função signo (com static char buffer[25])

Escreva um programa que leia o dia e mês de nascimento de uma pessoa. Com esses dados, invoque a função `signo` e indique ao utilizador o respetivo signo.

Corrija a função signo de modo a que a variável `buffer` não desapareça quando a função termina (usando `static`). Complete a função para os restantes meses do ano.

Tarefa única (para avaliação na aula seguinte (1.5 valores)): Manipulação de NIB

Escreva um programa que começa por invocar uma função menu. A função menu apresenta um menu no ecrã (com sucessivos `printf`), com as seguintes opções:

ler NIB
indicar o banco do NIB
indicar o número da conta (sem zeros à esquerda)
indicar os números de controlo (os dois últimos dígitos)
escrever o NIB com um espaço entre cada bloco (banco, agência, conta, controlo)
verificar o NIB (opcional)
sair do programa

As funções para manipular o NIB devem estar num ficheiro à parte (`nib.h` e `nib.c`).
O menu deve ser uma função, que dentro chama as outras funções. Depois de executar uma função, o menu é apresentado novamente, até que seja escolhida a opção 'sair do programa'. Para ler as entradas do menu, use algarismos ou letras.

Deixe para o fim a verificação do NIB. Para ter a avaliação completa basta ter as outras opções a funcionar.

Na Wikipédia encontra a informação sobre os NIB, em
`https://pt.wikipedia.org/wiki/N%C3%BAmero_de_Identifica%C3%A7%C3%A3o_Banc%C3%A1ria`