



**UAlg FCT**

UNIVERSIDADE DO ALGARVE  
FACULDADE DE CIÊNCIAS E TECNOLOGIA

# Programação Imperativa

Lição n.º 3

Programação com C

# Programação com C

- Problemas de programação.
- Decomposição funcional.
- Funções em C.
- Funções de teste.



# Problemas de programação

- Tipicamente, a tarefa de um programador é escrever programas para realizar determinadas tarefas, ou para resolver determinados problemas.
- Problema de hoje: escrever um programa C para calcular a nota final em *Programação Imperativa*, dada a nota da parte prática e a nota do exame.

# Problema da nota final

- A nota final é a média ponderada da nota da parte prática e da nota do exame, com pesos 30% e 70%, respetivamente.
- Mas se a nota do exame for menor que 8.5, a nota final é a nota do exame.
- As notas são expressas na escala de 0 a 20.
- A notas da parte prática e do exame são expressas com uma casa decimal.
- A nota final é expressa na forma de um número inteiro, obtido por arredondamento do resultado dos cálculos.

# Funções identificadas no enunciado

- A função para a média ponderada da nota da parte prática e da nota do exame.
- A função que se ocupa do caso em que a nota do exame é menor do que 8.5.
- A função que arredonda (para o número inteiro mais próximo) o resultado dos cálculos.
- A função de arredondamento é uma **função geral**, que podemos supor vir a ser útil noutros problemas.
- As outras são específicas deste problema.

# Média ponderada

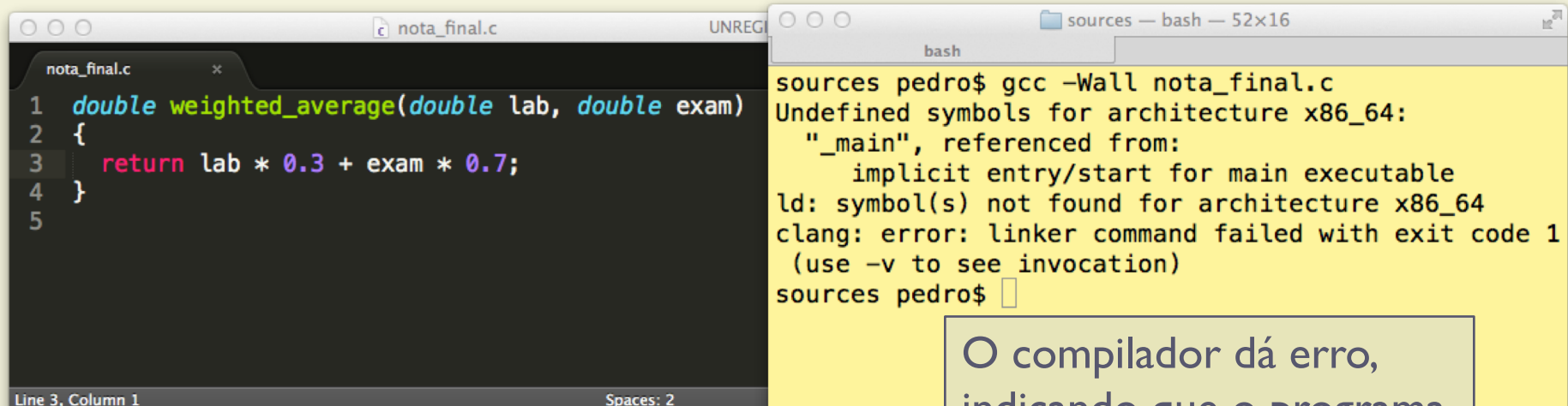
- Se **x** representar a nota da prática e **y** a nota do exame, a média ponderada desses dois valores, usando os pesos 30% e 70%, é dada pela expressão  $x * 0.3 + y * 0.7$ .
- As variáveis **x** e **y** denotam números reais, com parte decimal.
- Em C, os números reais são representados pelo tipo **double**.
- A função para a média ponderada terá dois argumentos de tipo **double** e o resultado também é de tipo **double**.

# Função `weighted_average`

- Observe:

```
double weighted_average(double lab, double exam)
{
    return lab * 0.3 + exam * 0.7;
}
```

- Usamos aqueles nomes `lab` e `exam` para deixar claro o significado dos argumentos.



The image shows a code editor window titled 'nota\_final.c' with the following code:

```
1 double weighted_average(double lab, double exam)
2 {
3     return lab * 0.3 + exam * 0.7;
4 }
5
```

Below the code editor is a terminal window titled 'sources — bash — 52x16'. It shows the command `gcc -Wall nota_final.c` being executed, resulting in the following error message:

```
sources pedro$ gcc -Wall nota_final.c
Undefined symbols for architecture x86_64:
  "_main", referenced from:
    implicit entry/start for main executable
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1
(use -v to see invocation)
sources pedro$
```

O compilador dá erro, indicando que o programa não tem uma função `main`.

# Função de teste

- Escrevamos uma função de teste para exercitar a função **weighted\_average**:

```
void test_weighted_average(void)
{
    double lb;
    double ex;
    scanf("%lf%lf", &lb, &ex);
    double z = weighted_average(lb, ex);
    printf("%f\n", z);
}
```

O compilador continuaria a dar erro, porque continua a faltar a função **main**.



# Especificadores de conversão

- A **cadeia de formato** contém um ou mais **especificadores de conversão**. Observe:

```
void test_int_format_string(void)
{
    int x;
    scanf("%d", &x);
    int y = x * x;
    printf("%d\n", y);
}
```

Ler: **%d**

Escrever: **%d**

```
void test_double_format_string(void)
{
    double x;
    scanf("%lf", &x);
    double y = x * x;
    printf("%f\n", y);
}
```

Ler: **%lf**

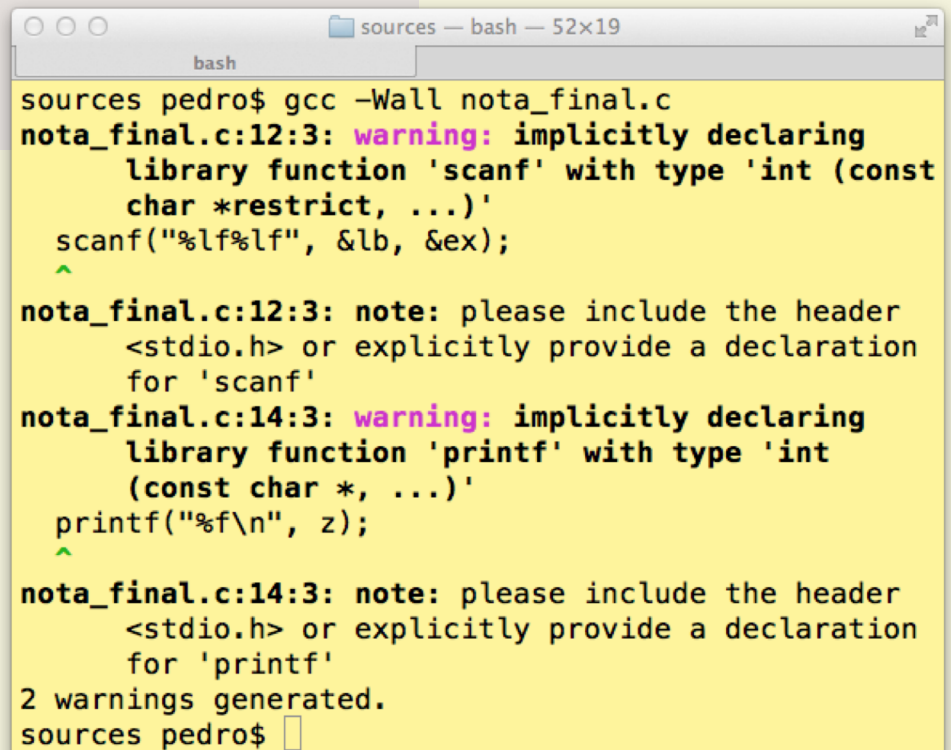
Escrever: **%f**

# Função main

- A função **main** chama a função de teste:

```
int main(void)
{
    test_weighted_average();
    return 0;
}
```

O compilador dá outro erro agora (na verdade, trata-se de um **warning**...) e sugere que incluamos o “header” `<stdio.h>`.



```
sources pedro$ gcc -Wall nota_final.c
nota_final.c:12:3: warning: implicitly declaring library function 'scanf' with type 'int (const char *restrict, ...)'
scanf("%lf%lf", &lb, &ex);
^
nota_final.c:12:3: note: please include the header <stdio.h> or explicitly provide a declaration for 'scanf'
nota_final.c:14:3: warning: implicitly declaring library function 'printf' with type 'int (const char *, ...)'
printf("%f\n", z);
^
nota_final.c:14:3: note: please include the header <stdio.h> or explicitly provide a declaration for 'printf'
2 warnings generated.
sources pedro$
```

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

```
double weighted_average(double lab, double exam)
{
    return lab * 0.3 + exam * 0.7;
}
```

```
void test_weighted_average(void)
{
    double lb;
    double ex;
    scanf("%lf%lf", &lb, &ex);
    double z = weighted_average(lb, ex);
    printf("%f\n", z);
}
```

```
int main(void)
{
    test_weighted_average();
    return 0;
}
```

Este é o programa completo.  
Tem uma função de cálculo, uma  
função de teste e a função **main**.  
À cabeça vem a diretiva **#include**  
**<stdio.h>**.

# Experimentando

- Compilamos e corremos na janela de comando:

```
sources pedro$ gcc -Wall nota_final.c
sources pedro$ ./a.out
10 12
11.400000
sources pedro$ ./a.out
15 18
17.100000
sources pedro$ ./a.out
17.2 14.5
15.310000
sources pedro$ ./a.out
14.8 7.1
9.410000
sources pedro$
```

De cada vez que corremos o programa, só fazemos uma experiência.

Note que o resultado é mostrado com 6 casas decimais.

# Experimentando repetidamente

- É simples: depois de escrever o resultado, chamamos a função de teste, de novo:

```
void test_weighted_average(void)
{
    double lb;
    double ex;
    scanf("%lf%lf", &lb, &ex);
    double z = weighted_average(lb, ex);
    printf("%f\n", z);
    test_weighted_average();
}
```

Paramos o programa,  
interrompendo-o, com  
**ctrl-C**.

```
sources pedro$ ./a.out
12.9 10.0
10.870000
8.5 12.7
11.440000
14.8 12.0
12.840000
^C
sources pedro$
```

# Experimentando repetidamente, melhor

- Em vez de interromper o programa à bruta, com **ctrl-C**, é melhor deixar o programa prosseguir, repetindo as contas, até acabarem os dados.
- Isto é, enquanto houver dados, o programa faz as contas; quando o programa “perceber” que não há mais dados, terminará imediatamente.
- Na consola, o fim dos dados é assinalado com **ctrl-Z** em Windows e com **ctrl-D** em Linux/MacOS.

Usamos **ctrl-C** para interromper um programa que disparatou ou um programa que chamámos por engano, não para fazer um programa sair de um ciclo.

# Ciclo de teste

- Observe com muita atenção:

```
void test_weighted_average(void)
{
    double lb;
    double ex;
    while (scanf("%lf%lf", &lb, &ex) != EOF)
    {
        double z = weighted_average(lb, ex);
        printf("%f\n", z);
    }
}
```

Eu terei dado **ctrl-D** para assinalar o fim dos dados, mas o **ctrl-D** não ficou visível na consola.

```
sources pedro$ ./a.out
12 15
14.100000
13.8 19.0
17.440000
10.1 19.9
16.960000
sources pedro$
```

# Função da nota exata, **grade**

- A média ponderada nem sempre dá a nota; só dá quando a nota do exame é maior ou igual a 8.5.
- Caso contrário, o resultado é a nota do exame.
- Observe:

```
double grade(double lab, double exam)
{
    return exam >= 8.5
        ? weighted_average(lab, exam)
        : exam;
}
```



# Função de teste para a nota exata

- Para controlo, incluímos também uma chamada à função **weighted\_average**:

```
void test_grade(void)
{
    double lb;
    double ex;
    while (scanf("%lf%lf", &lb, &ex) != EOF)
    {
        double v = weighted_average(lb, ex);
        printf("%f\n", v);
        double z = grade(lb, ex);
        printf("%f\n", z);
    }
}
```

É boa ideia mostrar os resultados das duas funções, porque sabemos que a segunda recorre à primeira.

# A nova função main

- A função **main** chama agora a nova função de teste.
- A anterior função de teste continua lá, mas comentada:

```
int main(void)
{
    // test_weighted_average();
    test_grade();
    return 0;
}
```

Tipicamente, as funções **main** são assim: chamam uma de várias funções de teste, estando as outras comentadas, para poderem facilmente ser reativadas, se necessário.

# Experimentando a nota exata

- Eis uma sessão de experimentação, usando a nova função **main**:

Confirmamos que nos casos em que a nota do exame é menor que 8.5, as duas funções dão resultados diferentes.

```
sources pedro$ ./a.out
14 10
11.200000
11.200000
16 8
10.400000
8.000000
16 8.4
10.680000
8.400000
16 8.5
10.750000
10.750000
19 6.2
10.040000
6.200000
sources pedro$
```

# Conclusão

- Já conseguimos calcular a nota exata, isto é, a nota calculada com toda a precisão.
- Falta calcular a nota final, arredondada.
- Note que as funções presumem que os valores dos argumentos fazem sentido, isto é, que são números reais entre 0.0 e 20.0, expressos com uma casa decimal, mas o programa não controla isso, e calcula cegamente.
- Aliás, se na função de teste fornecermos “lixo”, isto é, sequências de caracteres que não constituem números decimais, o programa estoirá ingloriamente nas funções de teste iterativas.