

ALGORITMOS E COMPLEXIDADE

GUIÃO DAS AULAS PRÁTICAS

**Joaquim João Estrela Ribeiro Silvestre Madeira
João Manuel de Oliveira e Silva Rodrigues**

**Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro**

Ano Letivo de 2019/2020

Programa das aulas práticas de Algoritmos e Complexidade

- A Linguagem de Programação C (1 aula)
- Análise Experimental da Complexidade de Algoritmos (4 aulas)
- Tipos de Dados Abstratos simples baseados em *arrays* e listas (2 aulas)
- Análise Experimental da Complexidade de Algoritmos Recursivos (2 aulas)
- Árvores Binárias ABP (2 aulas)
- Filas com Prioridade (1 aula)
- Grafos (2 aulas)

Material para as aulas práticas

- Os guiões e ficheiros necessários para as aulas práticas são disponibilizados no espaço de **E-Learning** da Universidade de Aveiro (<https://elearning.ua.pt/>).

Modo de Avaliação

- A nota final da unidade curricular é sempre calculada, em qualquer das épocas de exame, usando a nota do respetivo **exame escrito** – com um peso de **50%** – e a nota da **componente prática** (guiões e trabalhos entregues ao longo do semestre) – com um peso de **50%**; em ambas as componentes há uma **nota mínima de 8.0 valores (em 20)**.
- A nota da componente prática é calculada usando as notas de três trabalhos práticos – com um peso de 35% – e a nota das restantes aulas práticas – com um peso de 15%.
 - O **primeiro trabalho** prático tem um peso de 10% e é dedicado à análise da complexidade de algoritmos simples. Consiste na resolução dos guiões das aulas práticas 3 e 4 e da entrega dos guiões devidamente preenchidos.
 - O **segundo trabalho** prático tem um peso de 10% e é dedicado à implementação de tipos de dados abstratos. Consiste na resolução dos guiões das aulas práticas 5 e 6 e da entrega dos ficheiros de implementação dos tipos de dados abstratos.
 - O **terceiro trabalho** prático tem um peso de 15% e é dedicado à implementação de algoritmos sobre grafos. Consiste na resolução dos guiões das aulas práticas 12 e 13 e da entrega do correspondente ficheiro de implementação.
 - A resolução dos guiões das restantes aulas tem um peso de 15% na nota prática, com especial incidência nas aulas 7 e 8, sobre recursividade (peso de 7.5%), e nas aulas 9 e 10, sobre árvores binárias (peso de 7.5%). A avaliação é feita através da entrega dos guiões devidamente preenchidos e do ficheiro de implementação do tipo de dados abstrato árvore binária de pesquisa (abp.c).

AULA 0 – LINGUAGEM DE PROGRAMAÇÃO C

1. Usando um editor à sua escolha, crie um ficheiro com o programa seguinte (**simplest_1.c**). Use o seu compilador da linguagem C para compilar o ficheiro; analise eventuais mensagens do compilador. Execute o programa de aplicação resultante.

```
int main ( void )
{
    // O programa mais simples em C
}
```

Nota algum pormenor estranho neste programa?

Use a *flag* de compilação **-Wall** que lhe permite obter eventuais avisos de compilação.

Experimente compilar o ficheiro usando diferentes normas da linguagem C – **-std=c11** ou **-std=c99** ou **-std=c90** – e analise eventuais mensagens do compilador.

Modifique o ficheiro de acordo com o exemplo seguinte. Compile e execute o programa usando as diferentes normas da linguagem. Analise eventuais mensagens do compilador e o resultado do programa.

```
#include<stdio.h>

int main ( void )
{
    printf ( "%ld\n", __STDC_VERSION__ );

    return 0;
}
```

2. Crie um ficheiro com o programa seguinte (**hello_world.c**). Compile e execute o programa de aplicação resultante; analise o resultado.

```
#include<stdio.h>

int main ( void )
{
    puts ( "Hello World!" );

    return 0;
}
```

Crie uma outra versão do exemplo (**hello_X.c**), que peça o nome próprio e o apelido do utilizador e escreva, por exemplo, “**Hello Joaquim Madeira!**”.

Experimente usar as funções **printf()** e **puts()**, e **scanf()**, **gets()** e **fgets()** – analise eventuais mensagens do compilador.

Modifique o exemplo, para ler o nome próprio e o apelido do utilizador a partir da **linha de comandos**.

3. Desenvolva um programa que escreva uma **tabela** com os **quadrados** e as **raízes quadradas** dos primeiros números naturais. O utilizador deve indicar quantas linhas tem a tabela.

Formate as colunas da tabela de modo apropriado; as colunas da tabela deverão ter um cabeçalho.

O que é necessário fazer para poder usar a função `sqrt()`?

4. Modifique o programa anterior para que escreva uma **tabela** com sucessivos valores do **seno** e do **cosseeno**. O utilizador deve indicar o menor valor e o maior valor do ângulo, e o espaçamento entre sucessivos valores intermédios. Por exemplo:

ang	sin(ang)	cos(ang)
0	0.0000000000	1.0000000000
5	0.0871557427	0.9961946981
10	0.1736481777	0.9848077530
15	0.2588190451	0.9659258263
20	0.3420201433	0.9396926208

Use um **menor número de casas decimais** em cada coluna.

Modifique o programa para que a tabela seja escrita num **ficheiro**.

5. Desenvolva um programa que liste o número de bytes usados para representar os tipos primitivos da linguagem C, usando a função `sizeof()`.

Por exemplo, numa arquitetura de **32 bits** obteve-se:

```
sizeof(void *) ..... 4
sizeof(void) ..... 1
sizeof(char) ..... 1
sizeof(short) ..... 2
sizeof(int) ..... 4
sizeof(long) ..... 4
sizeof(long long) ... 8
sizeof(float) ..... 4
sizeof(double) ..... 8
```

Compare os resultados acima com aqueles que obtém numa arquitetura de **64 bits**.

6. Considere o programa em **Java** listado na página seguinte, que apresenta um simples exemplo de utilização de *arrays*.

Usando a **linguagem C**, desenvolva um programa equivalente com o mesmo tipo de funções.

```
/*
Crie um programa em C equivalente a este em Java.
*/

public class ProgA {

    public static void main(String[] args) {
        int[] a = {31,28,31,30,31,30,31,31,30,31,30,31};
        printArray("a", a);

        int[] b = new int[12];
        cumSum(a, b);
        printArray("b", b);
    }

    public static void printArray(String s, int[] a) {
        System.out.println(s + ":");
        for (int i=0; i<a.length; i++) {
            System.out.print(a[i]+" ");
        }
        System.out.println();
    }

    public static void cumSum(int[] a, int[] b) {
        int c = 0;
        for (int i=0; i<a.length; i++) {
            c += a[i];
            b[i] = c;
        }
    }
}
```

Atenção, na linguagem C:

- Uma função ter de ser definida (ou declarada) antes da sua primeira invocação.
- Um *array* é declarado de modo diferente.
- Quando um *array* é argumento de uma função é, também, necessário passar como argumento o seu tamanho.

