

IAED

2019

email p/problemas da disciplina: iaed-lagus@tecnico.ulisboa.pt

email p/problemas técnicos: reis.santos@tecnico.ulisboa.pt

Dúvidas → preferencialmente no Slack

Avaliação

Exame (sobre algoritmos) → 30%; mínimo 8 valores

Avaliação nos labs a partir da 3ª semana (segunda ou terça) → contém as 4 melhores dos 5 → mínimo 8 valores; 30%.

Projetos - soma de 2 entregas ponderada pelo teste prático → 40%; mínimo 8 valores

1º projeto → estruturas e vetores → entrega a 24/03 (20h)

2º projeto → ponteiros → entrega a 14/04 (20h)

os projetos são avaliados de 0 a 10 e o teste prático de 0 a 20

Algoritmo

Sequência de operações que permite resolver um problema

Para melhorar o desempenho do algoritmo é necessário, em geral, torná-lo mais complexo:

- mais operações, mais memória
- estruturas de dados mais elaboradas

Operações lógicas

23/02/2023

and &&

or ||

not !

Todo o código tem de estar dentro de uma função.

'a' → carácter

"a" → cadeia de caracteres

podemos aceder remotamente
ssh sigma.tecnico.ulisboa.pt

```
int a;  
a = 12;
```

a = 12

a = "abc"

a = 1.2;

da warning
→ guarda o ponteiro
da cadeia de
caracteres

→ não há problema
seguir a parte real

em C todas as variáveis não declaradas são consideradas inteiras e os tipos das funções também

programa: indent hello.c → arruma o código
no IDE também há essa funcionalidade

```
if (a > b)
```

```
    b = a;
```

← a = a + 1; assim esta instrução não está dentro do if

```
if (a > b) {
```

```
    b = a;
```

```
    a = a + 1; assim já se encontra dentro
```

```
}
```

%d decimal int

%o octal int

%X %x hexadecimal int

%s string

%c char → caracter

%g %e %f real

24/02/2023

comentários em C

começam com: /*

terminam com: */

~~~~~;~~~~~;~~~~~; ← terminador  
~~~~~,~~~~~,~~~~~ ← separador

2 * 3 → 6.0

5.19 * (fahr - 32)
real int
~~~~~  
real

%3.0d → 0 dígitos à direita  
1 dígito para ponto  
2 dígitos à esquerda

%6.1d → 1 dígito à direita  
1 dígito para ponto  
4 dígitos à esquerda

não colocar constantes literais no meio do código

$\pi$  → constante

3.1415... → constante literal

**DEFINIR CONSTANTES**

#define



28/02/2023

$1 \& 2 \rightarrow \emptyset$  (FALSE)

decimal 12

OCT 1 014

hexadecimal oxide

binário ob 1100

1 → 0b01

$2 \rightarrow 0010$      $8 \rightarrow \text{and}$

$$0500 \rightarrow 0$$
$$\begin{array}{r} \text{ab } 11 \rightarrow 3 \\ \hline \end{array}$$
$$0b100 < 2: 0b10000 = 48$$

(x4)

$$(x^2)^2$$
$$cb1100 \gg 2 = cb11 = 3$$
$$\text{ob } 1(x) \gg 3 = \text{ob } 1 = 1$$

\$ gcc ex01.c -o ex01

\$/excl

Seche "6ln2ln5"  $\rightarrow$  exol

6 2 5 input

6 output (o.m.a.c)

\$ 1proj \leq \text{input.}

buffering { line  
block  
no

UTF-8

ASCII 107  $2^7 = 128$

1	C	P	2
---	---	---	---

180-LATIN

1	1	1	1	4x
---	---	---	---	----

$$2^2 \quad 2^2 \quad 2^2 \quad 2^2 \quad 2^2$$

int a = 'T', 10085 → C

a: crd('T') → python

em C

int x = 12

 $x + y$ 

é válido

```
printf("%s\n", msg);
```

no ~~trabalho~~ o que significa

$$\wedge V \wedge V \rightarrow \wedge V$$
$$\wedge V \wedge V \wedge V \wedge V \rightarrow \wedge V \wedge V$$
$$EGF = -1$$

↳ End of File

(no terminal unix =  $C+1-D$ )

Windows ^z \n

no início da linha

=  $\rightarrow$  operador menos prioritário em C

while (c = (getchar() != EOF)) resto caso comparando as duas

0.5



# Contagem de Listas

27/02/2022

Capitamentos  
da Kika

## Incremento da variável contador

contador=0

### • ciclo while

↳ onde podemos usar  $i++$  ou  $i+=1$

ou

### • ciclo for

contador=0

teste:  $v \geq 0$

incremento:  $++$  contador

## → Utilizar:

•  $a++$  ou  $++a$  é diferente

$a++ \leq 3$

↳ soma depois de testar  $a \leq 3$

$++a \leq 3$

↳ soma antes de testar  $a \leq 3$

## Resumidamente

usar incrementos

prefixamente ou

suffixamente influencia

o programa

float = %f

scanf = (%f, &v)

double = %lf

↳ preciso escrever esse símbolo antes da  
variável para o programa conseguir ler

## CONTAGEM DE PALAVRAS

01/03/2023

C=48

C='0'

<ctype.h>

isnum(c) → número

isalpha → letra

isupper → maiúscula

islower → minúscula

isalnum →

isblank → caracter branco (inclui \n)

" " → string vazia

entre " " tem de lá

estar alguma coisa

## OPERAÇÕES SOBRE ARQUIVOS

read → "r"

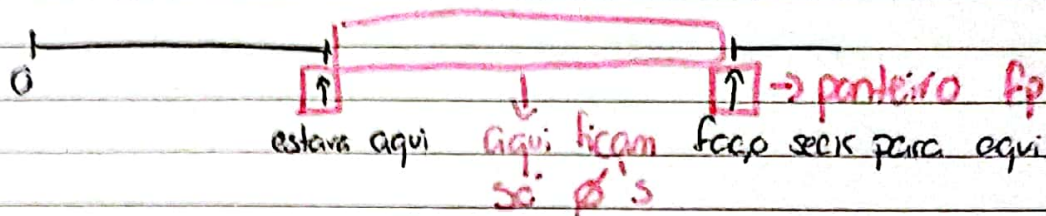
write → "w"

append → "a"

"w+" → abrir para ler e escrever  
ver slide.



lock(fp, des), =



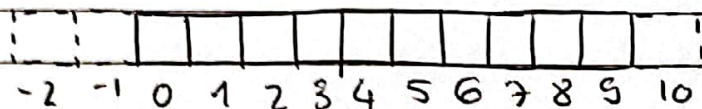
FILE \*fp  
diz que é um ponteiro

fprintf("teste.txt") → imprime o último erro que deu

não esquecer de fechar o ficheiro no fim

# vectores

sequências do mesmo tipo



tab[DIM] é um inteiro

tab é um ponteiro

scanf("%d", &i);  
tab[i] if (i < 0 || i >= DIM) error();

for (i = 0; i < DIM; i++)  
scanf("%d", &tab[i]);

printf("%s?c\n", str, str[2]);  
%x | n° ASCII  
%d

"%100s" → limite, lê até 100 caracteres  
"%[^\\n]" → lê exceto \\n  
"%[a-zA-Z]" → lê algarismos  
"%[aeiou]" → lê vogais

\$ la.out um dois três  
argc[0] argc[1] argc[2] argc[3] argc[4]  
10

3
2
1
0

a=1;  
f(1,2,3);  
b=2;

printf("%d %s %d\n", 12, "abc", 'c');

"%d %s %d\n"

03/03/2023

"c"
"abc"
12



nos projetos todas as warnings são consideradas erros

strcmp → compara strings 0 se não forem iguais 1 se forem

Comando **man** no terminal permite aceder aos manuais

man 1	comandos terminal
man 2	chamadas de sistema
man 3	rotinas de biblioteca

→ apartamentos da Beatriz ←

ex: argumentos do programa

(...)

→ nº argumentos

→ variáveis ambiente

int main(int argc, char \*argv[], char \*envp[])

↳ ponteiro para lista argumentos

• ordem em que argumentos são chamados: f(1,2,3)

↳ 3,2,1, chama função f

## Resumo

- strings são vetores de caracteres com um '\0' no fim
- formato escrita e leitura para strings: %s
- vetores copiados posição a posição

## string.h

- strcmp (compara strings)
- strncmp (compara até char n)
- strcpy (copia strings)
- strlen (devolve tamanho da string)
- strchr (devolve posição do carácter na string)
- strtok (divide string por separadores: split)
- strtol (string para inteiro 2 ≤ base ≤ 35, erro e overflow)

## ~~Funções~~

- Definição

<tipo retorno> <nome> (<declaração argumentos>);

{

<declarações>

<instruções>



## → instrução return

- permite retornar um valor da função para uma outra função onde foi invocada

**return <expressão>;**

- valor expressão convertido para o valor de retorno da função
- função termina ao executar instrução return

**\*main function always comes last**

## Passagem por valor e por referência

- parâmetros actuais copiados para variáveis temporárias quando função é executada
- função não tem acesso aos parâmetros (só as cópias) e não os pode alterar
- diferente da "passagem por referência" (Pascal)
- Exceção: se o parâmetro actual é uma tabela, não é efectuada a cópia da tabela  
↳ se a função alterar o conteúdo da tabela, tem efeitos fora da função

**ex: Passagem por valor**

```
int potencia(int base, int n)
{
```

```
    int i, p=1;
    for (i=1; i<=n; i++)
        p=p*base;
    return p;
}
```

```
int potencia(int base, int n)
{
```

```
    int p;
    for (p=1; n>0; n--) modificação do
        p*=base; valor n não é
    return p; passado para fora
               da função
```

**ex: Passagem por referência (passar ponteiro)**

```
int potencia(int base, int n)
{
```

```
    int i, p=1;
    for (i=1; i<=n; i++)
        p=p*base;
    return p;
}
```

```
int main()
{
```

```
    int base, ex;
    scanf("%d %d", &base, &ex); → leitura formatada do input
    printf("%d %d %d\n", base, ex, potencia(base, ex)); → passagem por referência
    return 0;
}
```



ex: Passagem tabelas → indicamos que a função recebe um vetor de inteiros

**void** inicializar-tabela(int tabela[], int tamanho)

{ → função não retorna nada

int i;

for (i=0; i < tamanho; i++)

tabela[i] = 0;

}

int main()

{

int vec[100];

inicializar-tabela(vec, 100);

return 0;

}

06/03/2023

f(int vec[]);

f(int \*vec);

no caso das strings: nunca esquecer do espaço para o \0.

assert

↳ faz uma verificação se falhar sai da função

## Estruturas

07/03/2023

Permite definir estruturas de dados sofisticadas, as quais possibilitam a agregação de diferentes tipos de declarações

typedef struct aluno{

int numero;

char nome[10];

float nota;

} Aluno, \*ALUNO

struct aluno a1;

Aluno a2;

ALUNO a3 = &a1;

09/03/2023

if ("abc" == "abc")

↳ compara ponteiros

para comparação de strings strcmp()

devolve 0 se iguais





```
if (!strcmp(____, ____))  
    se 1 é verdadeiro  $\rightarrow$  strcmp() = 0
```

# Algoritmo

Procedimento computacional bem definido que aceita uma dada entrada e produz uma dada saída.

Ferramenta para resolver um problema computacional bem definido

- calcular a média de um conjunto de valores
- ordenação de seqüências de valores
- caminhos mais curtos em grafos orientados
- etc.

## Análise de Algoritmos

### → Eficiência

- escolha do algoritmo certo!
- vamos aprender a mapear um algoritmo numa determinada classe de eficiência/complexidade

### → Medidas de eficiência e complexidade

- tempo (tempo de execução) e espaço (memória)

todos os algoritmos que vamos ver nesta cadeira são in-memory 10/03/2013  
vamos ver se são ou não in-place (se precisam de memória auxiliar)  
e vamos ver se são estáveis (chaves iguais)

```
typedef struct aluno Item;
```

```
#define KEY(A) A.name
```

```
#define less(A,B) strcmp(KEY(A), KEY(B))
```

```
#define KEY(A) A.nota
```

```
#define KEY(A) A.numero
```

temos a estrutura

```
struct Aluno {
```

```
    char nome[100];
```

```
    int numero;
```

```
    float nota;
```

```
}
```

e depois definimos



lst.sort() → passa o ponteiro para o vetor e ordena

lst = sorted(lst)

selection sort

$$\underbrace{O(n^1) \quad O(n^2)}_{\theta(n^2)}$$

## Ex 6

análise assintótica

14/03/2023

②  $\Omega(1) \rightarrow$  melhor caso

$O(\log_2 N) \rightarrow$  pior caso  
Ex: algoritmos sempre lida em 2

$$\begin{matrix} O(1) \\ \Omega(1) \\ \theta(1) \end{matrix}$$

③  $\theta(N) - O(N)$   
     $\searrow \Omega(N)$

④  $\theta(M \log N)$

⑤  $\theta(N^2)$

⑥  $\Omega(1)$   
     $O(N)$

$i = 1/2$  percorre  $\frac{n}{2}$

## Ex 7

17/03/2023

$v[] = \{38, 10, 5, 48, 20, 9, 40, 25\}$

bubble: compara 2 a 2

10, 5, 38, 20, 9, 40, 25, 48  
5, 10, 20, 9, 38, 25, 40, 48  
5, 10, 9, 20, 25, 38, 40, 48  
5, 9, 10, 20, 25, 38, 40, 48  
5, 5, 10, 20, 25, 38, 40, 48

faltam 2 entradas q confirmam que os outros 2 números estão no sítio

selection:

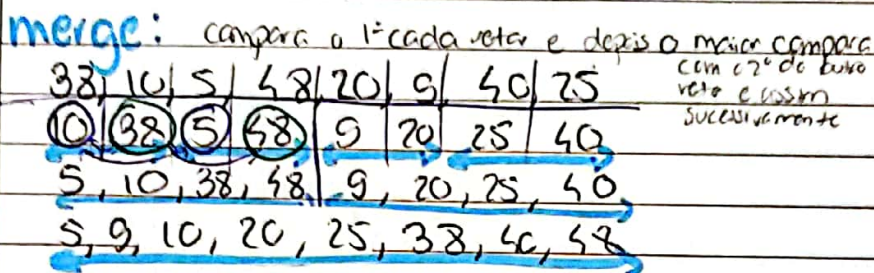
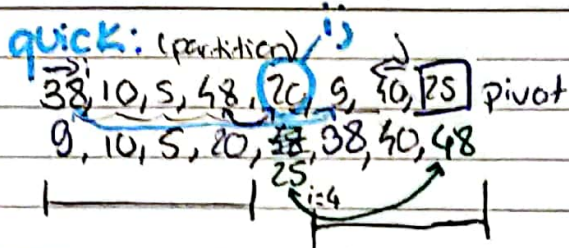
10, 38, 5, 48, 20, 9, 40, 25  
5, 10, 38, 48, 20, 9, 40, 25 só na  
5, 10, 38, 48, 20, 9, 40, 25 últimos itens  
5, 10, 20, 38, 48, 9, 40, 25 é que tenho o  
5, 9, 10, 20, 38, 48, 40, 25 vetor totalmente  
5, 5, 10, 20, 38, 40, 48, 25 ordenado  
5, 9, 10, 20, 25, 38, 40, 48

selection: troca de menor

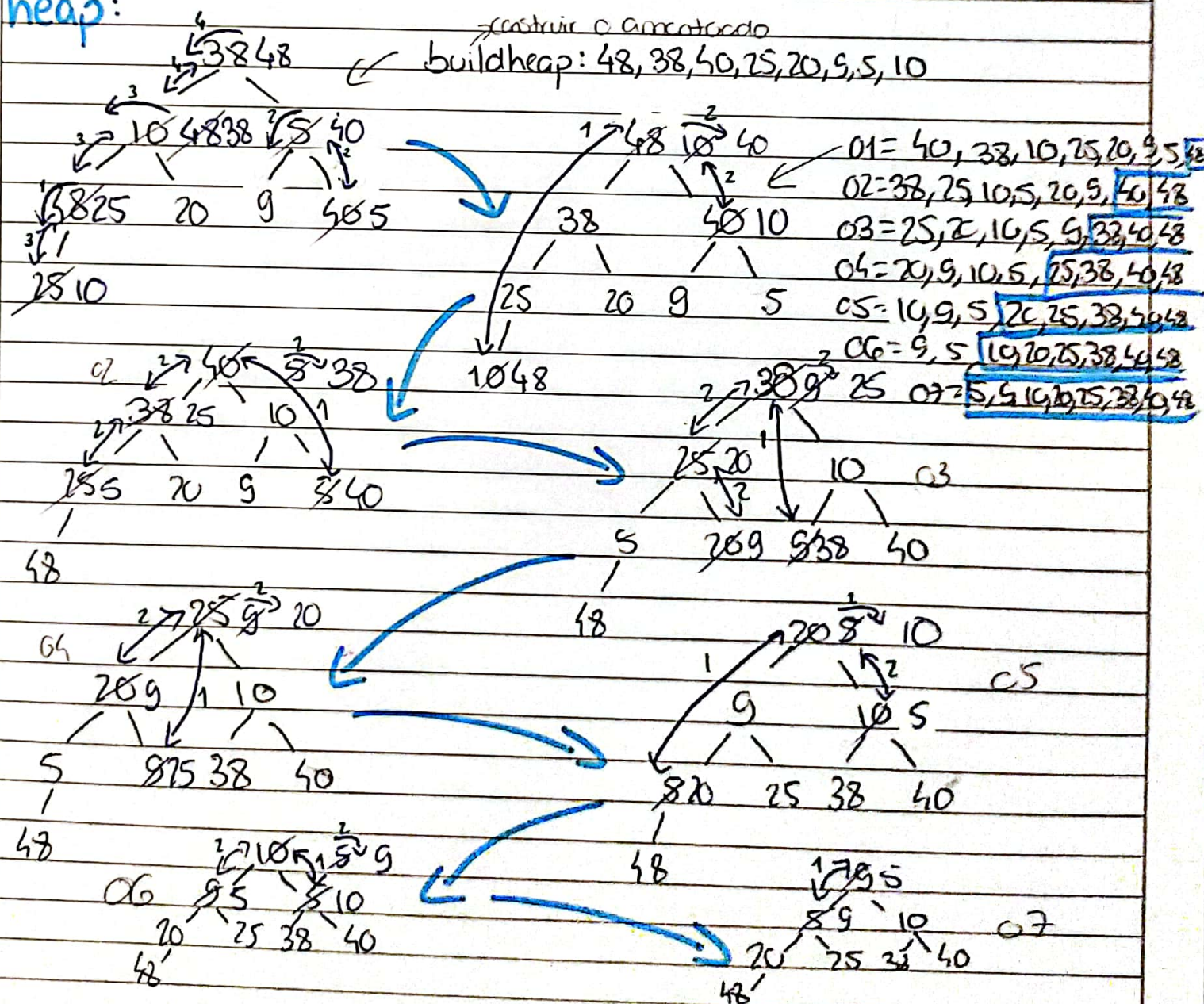
5, 10, 38, 48, 20, 9, 40, 25  
5, 9, 38, 48, 20, 10, 40, 25  
5, 9, 10, 48, 20, 38, 40, 25  
5, 9, 10, 20, 48, 38, 40, 25  
5, 9, 10, 20, 25, 38, 40, 48

⑦ faltam 2 entradas que confirmam que os 2 nº seguintes já estão no sítio





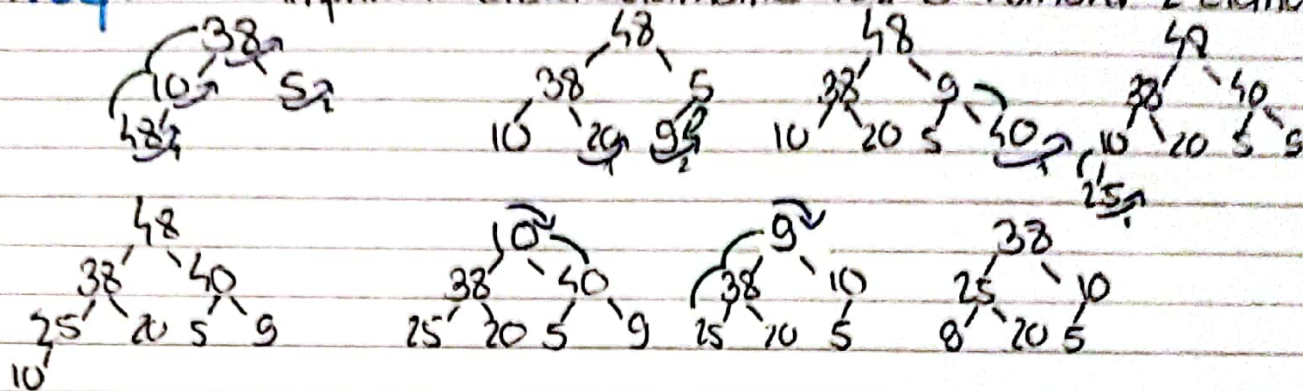
**heap:**





Príog:

imprimir ordenadamente v[] e remover 2 elementos



res: 38,25,10,9,20,5

no gcc

17/03/2023

char freq[100] = {1, 2, 3}; inicializa os 1<sup>os</sup> 3 elementos com os valores e os outros a zero.

## Ponteiros e Tabelas

pontero → variável que contém um endereço de outra variável

int x = 10;

int \*px = &x;



inicializo px com o endereço de x (operador &)

declaração de um ponteiro

char buf[100]; \*ptr = buf;

ptr = 1

\*buf

ptr[5];

\*(ptr + 5)

\*(buf + 5)

int v[10] = {5, 2, 3, 5, 2, 5, 4, 4, 4, 4}; \*ptr = v;  
 \*ptr++ = 5;  
 \*++ptr = 5;  
 (\*ptr)++;  
 ++(\*ptr);



# Ficha 1 exemplo

I.

```
#include <stdio.h>
int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    for (i = 1; i <= k; i++) {
        printf("%d\n", n * i);
    }
    return 0;
}
```

II.

1. A função `float c(int a, int b)` faz a média dos números ímpares que se encontram entre `a` e `b`.

$$2. \frac{11+13+15+17}{4} = \frac{56}{4} = 14$$

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

```
int main() {
```

```
    int c, soma;
```

```
    char men;
```

```
    scanf("%s", &men);
```

```
    c = getchar();
```

```
    while (c != EOF) {
```

```
        if (c >= '0' && c <= '9') {
```

```
            soma += (c - 48);
```

```
        }
```

```
        c = getchar();
```

```
    }
```

```
    printf("%d", soma);
```

```
    return 0;
```

```
}
```

01a 154

14



# Quiz 2 example

①

```
#include <stdio.h>
double polyval(double pol[], int size, double x)
```

```
    int i;
    double res = 0;
    for (i = 0; i < size; i++) {
        res += (pol[i] * pow(x, i));
    }
```

```
    return res;
}
```

②

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

```
int moda()
```

```
    int v[MAX], i, m[MAX], j, max; max = 0;
```

```
    for (i = 0; i < MAX; i++) {
```

```
        scanf("%d", &v[i]);
```

```
        if (v[i] < 0) break;
```

```
    }
```

```
    for (i = 0; v[i] >= 0; i++) {
```

```
        m[i] = 0;
        for (j = 1; v[j] >= 0; j++) {
```

```
            if (v[j] == v[i]) m[i]++;
        }
```

```
    }
```

```
    for (i = 0; v[i] >= 0; i++) {
```

```
        if (m[i] > max) max = m[i];
    }
```

```
    return i;
```

```
}
```

$v = \{0, 0, 0, 0, 0, \dots, 0\}$

2

$v[2]++$

$v = \{0, 0, 1, 0, 0, \dots, 0\}$

6

$v[6]++$

$v = \{0, 0, 1, 0, 0, 0, 1, \dots, 0\}$