

UNIVERSIDADE DA BEIRA INTERIOR

Programação – LEI

1º Semestre

Exame Época Normal

Resolução

2024/2025

1.

Escreva uma **expressão lógica** em linguagem C para as seguintes condições:

a) o valor da variável do tipo real **A** **pertence** ao intervalo **[-50, 50[**

$A \geq -50 \ \&\& \ A < 50$

b) o valor da variável do tipo inteiro **B** **é** um número **ímpar** e **pertence** ao conjunto **{10, ..., 99 }**

$B \geq 10 \ \&\& \ B \leq 99 \ \&\& \ B \% 2 == 1$

Escreva uma **instrução de atribuição** em linguagem C para cada uma das seguintes acções:

c) a variável **B** recebe a **soma** dos **dois últimos algarismos** do valor da variável do tipo inteiro **A** (exemplo: $A = 23542 \Rightarrow B = 6$, pois $6 = 4 + 2$)

$B = A \% 10 + (A / 10 \% 10);$

$B = A \% 10 + (A \% 100 / 10);$

d) a variável **P** recebe o valor **-5** se o valor da variável **N** for **par** e o valor **15** se **N** for **ímpar** (**P** e **N** são variáveis do tipo inteiro)

$P = (N \% 2) * 20 - 5;$

Supondo que **X = 5**, **Y = -2** e **Z = 2** (**X**, **Y** e **Z** são variáveis do tipo inteiro), indique a ordem de cálculo dos operadores e determine o valor de cada uma das seguintes expressões. Apresente todos os cálculos.

e) $Y \underset{1}{*} \underset{2}{(5 - Z)} \underset{3}{\geq} \underset{4}{(Z + X)} \underset{5}{\%} \underset{3}{3}$

ordem de cálculo: **2, 4, 1, 5, 3**

valor: **falso**

$-2 * (5 - 2) \geq (2 + 5) \% 3 = (2)$

$-2 * 3 \geq (2 + 5) \% 3 = (4)$

$-2 * 3 \geq 7 \% 3 = (1)$

$-6 \geq 7 \% 3 = (5)$

$-6 \geq 1 = (3)$

falso

f) $X \underset{1}{+} \underset{2}{(2.5 - Y)} \underset{3}{*} \underset{4}{(25 / (X * Z))}$

ordem de cálculo: **2, 5, 4, 3, 1**

valor: **14.0**

$5 + (2.5 - -2) * (25 / (5 * 2)) = (2)$

$5 + 4.5 * (25 / (5 * 2)) = (5)$

$5 + 4.5 * (25 / 10) = (4)$

$5 + 4.5 * 2 = (3)$

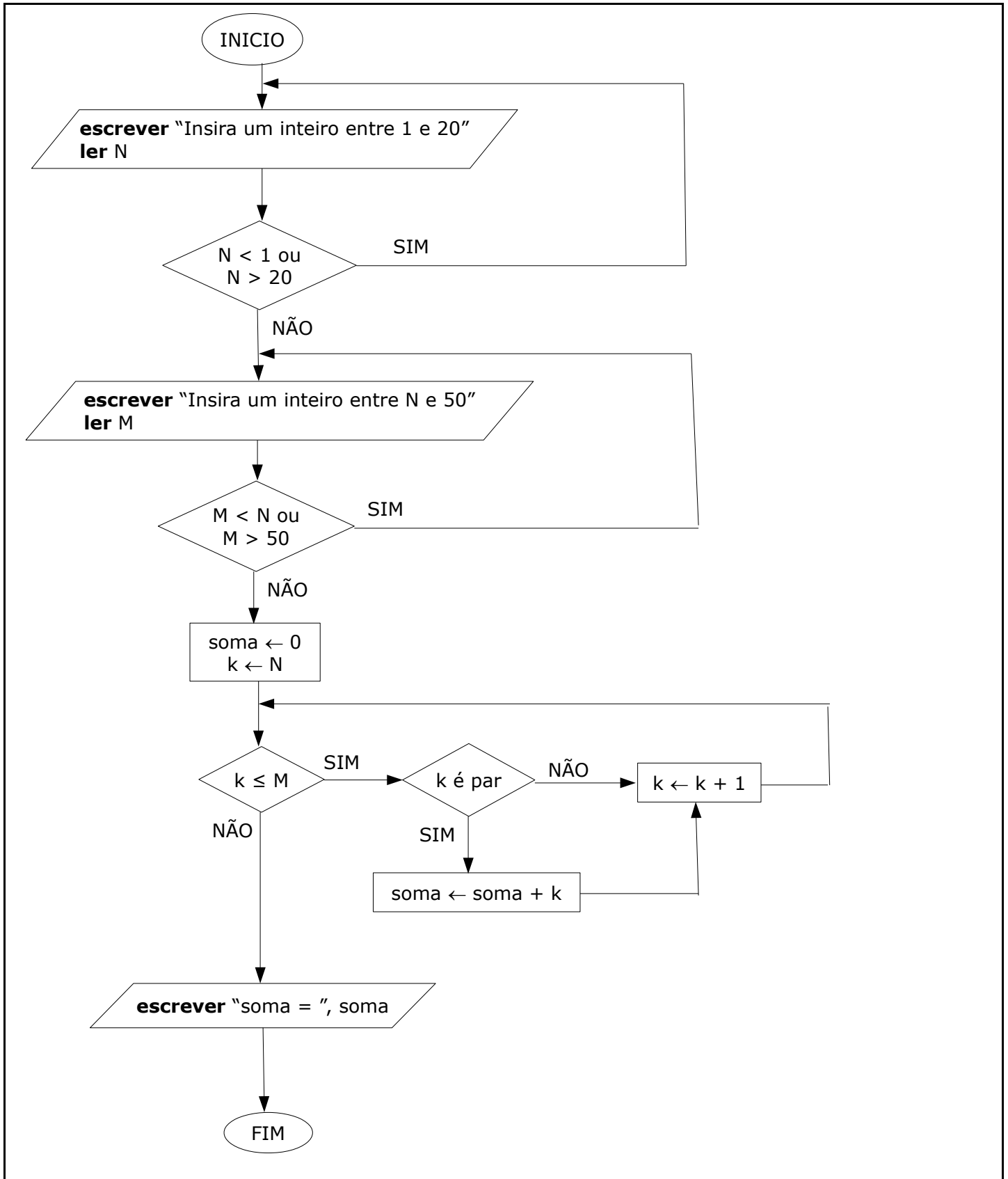
$5 + 9.0 = (1)$

14.0

2.

Construa um algoritmo (**sem usar arrays**), através de um fluxograma, que:

- **peça** ao utilizador e **insira/leia** um número **inteiro N** com valor **entre 1 e 20** ($1 \leq N \leq 20$)
- **peça** ao utilizador e **insira/leia** um número **inteiro M** com valor **entre N e 50** ($N \leq M \leq 50$)
- **determine** a **soma** dos números **pares entre N e M**, inclusivé
- **mostre** o **resultado obtido** (soma dos números pares).



3.

Construa um **programa em C** (**sem usar arrays**) que:

- **peça** ao utilizador e **insira/leia** um número **inteiro N** com valor **entre 1 e 20** ($1 \leq N \leq 20$)
- **peça** ao utilizador e **insira/leia** um número **inteiro M** com valor **entre N e 50** ($N \leq M \leq 50$)
- **determine** a **soma** dos números **pares entre N e M**, inclusivé
- **mostre** o **resultado obtido** (soma dos números pares).

```
#include <stdio.h>

int main()
{
    int M, N, k, soma;
    do{
        printf("Inserir um inteiro entre 1 e 20: ");
        scanf("%d", &N);
    }while(N < 1 || N > 20);

    do{
        printf("Inserir um inteiro entre %d e 50: ", N);
        scanf("%d", &M);
    }while(M < N || M > 50);

    soma = 0;
    for(k = N; k <= M; k++)
        if (k % 2 == 0)
            soma = soma + k;
    printf("Soma = %d\n", soma);
}
```

4.

Implementar um **subprograma** em C que dados um array (de 1 dimensão) **X** com **N** números reais (X e N são parâmetros do subprograma), **determine e devolva como resultados** a **quantidade** (número inteiro) de números **positivos de X** e a **média aritmética** (número real) dos números **positivos de X**.

NOTA: caso o array X não contenha números positivos, considerar que o valor da média é 0 (zero).

```
// devolve pelo return a quantidade de positivos e pelo cabeçalho o valor da média
```

```
int quantMedia (float X[], int N, float *media)
```

```
{
```

```
    int quant, k;
```

```
    float soma;
```

```
    quant = 0;
```

```
    soma = 0;
```

```
    for (k = 0; k < N; k++) {
```

```
        if (X[k] >= 0) {
```

```
            quant = quant + 1;
```

```
            soma = soma + X[k];
```

```
        }
```

```
    }
```

```
    if (quant > 0)
```

```
        *media = soma / quant;
```

```
    else
```

```
        *media = 0;
```

```
    return quant;
```

```
}
```

5.

Considere as seguintes declarações de variáveis:

int *V, **W, *X;

e que **sizeof(int) = 4** e **sizeof(int *) = 8**.

(esquema de um bloco de memória)

	...	
100500	110168	V
	...	
100700	110164	X
	...	
110160	90	
110164	800	
110168	700	
110172	600	
110176	50	
110180	40	
	...	
110500	100500	w
	...	

NOTA: se não existir resposta, indicar com **ERRO**

Usando os valores contidos no esquema de um bloco de memória dado ao lado, indique, justificando, os valores de cada uma das seguintes expressões:

- a)** $*(V + 2)$ = $*(110168 + 2 \times \text{sizeof(int)})$ = $*(110176)$ = **50**
- b)** $X - 3$ = $110164 - 3 \times \text{sizeof(int)}$ = **110152**
- c)** $\&V[2]$ = **110176**
- d)** $*(X - 1)$ = $*(110164 - 1 \times \text{sizeof(int)})$ = $*(110160)$ = **90**
- e)** $**W + 4$ = $700 + 4$ = **704**
- f)** $W + 4$ = $100500 + 4 \times \text{sizeof(int*)}$ = **100532**
- g)** $*(W + 3)$ = $*(110168 + 3 \times \text{sizeof(int)})$ = $*(110180)$ = **40**
- h)** $*W - 2$ = $110168 - 2 \times \text{sizeof(int)}$ = **110160**
- i)** $W[0]$ = **110168**
- j)** $X[1]$ = **700**

6.

Considere um ficheiro de texto de nome "**dados.txt**", em que **cada linha** deste ficheiro contém dois números: **um** número **real** e **um** número **inteiro**, por esta ordem. Implemente um **programa em C** que realize as seguintes acções, pela ordem indicada:

- 1º) construa** um **array 1D X** com os **números reais** contidos no ficheiro de texto "**dados.txt**", usando **gestão dinâmica de memória**,
- 2º) determine** o **maior** elemento e o **menor** elemento do array X, e
- 3º) guarde** os valores do **maior** e do **menor** obtidos no passo 2, no ficheiro de texto "**saida.txt**".

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

int main()
{
    float *X, num1, maior, menor;
    int tamX, num2, k;
    FILE *f;
    // 1º)
    f = fopen("dados.txt", "r");
    tamX = 0;
    X = (float*) malloc(tamX * sizeof(float));
    while(fscanf(f, "%f%d", &num1, &num2) == 2) {
        tamX = tamX + 1;
        X = (float*) realloc(X, tamX * sizeof(float));
        X[tamX-1] = num1;
    }
    fclose(f);
    // 2º)
    if (tamX > 0) {
        maior = X[0];
        menor = X[0];
    }
    for (k = 1; k < tamX; k++) {
        if (X[k] > maior)
            maior = X[k];
        else
            if (X[k] < menor)
                menor = X[k];
    }
    // 3º)
    f = fopen("saida.txt", "w");
    fprintf(f, "Maior = %f\n", maior);
    fprintf(f, "Menor = %f\n", menor);
    fclose(f);
}
```

7.

Implementar uma **função recursiva** em C que dados um **array 1D A** com **N** ($N \geq 1$) números **inteiros**, e um **número inteiro K** (com A, N e K parâmetros da função), **determine** quantos **elementos** do **array A** têm **valor** igual ao valor de **K**.

```
int quantIguais (int *A, int N, int K)
{
    int quant;
    // caso base/terminal
    if (N == 0)
        return 0;
    // caso geral
    quant = quantIguais(A, N-1, K);
    if (A[N-1] == K)
        return quant + 1;
    else
        return quant;
}
```

8.

Implementar um **subprograma** em C que dados um **array 1D A** com **N** números **inteiros** (A e $N > 0$ são parâmetros do subprograma), **determine e devolva o segundo maior elemento negativo do array A, caso exista** (se não existe, deve devolver um valor positivo).

NOTA: não pode usar outros arrays nem ficheiros, e deve percorrer o array o número mínimo de vezes.

Exemplo: $A = [2 \ -4 \ -9 \ 34 \ -12 \ 25 \ 76 \ -54] \Rightarrow$ segundo maior negativo = **-9** (maior = -4)

```
int segundoMaiorNegativo (int *A, int N)
```

```
{
```

```
    int maior, segMaior, k;
```

```
    k = 0;
```

```
    // encontrar o primeiro valor do array negativo
```

```
    maior = 1;
```

```
    while (k < N && maior == 1)
```

```
    {
```

```
        if (A[k] < 0)
```

```
            maior = A[k];
```

```
            k = k + 1;
```

```
    }
```

```
    // determinar o segundo maior valor negativo
```

```
    segMaior = maior; // mesmo que maior = 1
```

```
    while (k < N) // continuar com k com o valor anterior e não regressar ao inicio do array
```

```
    {
```

```
        if (A[k] < 0) {
```

```
            if (A[k] > maior) {
```

```
                segMaior = maior;
```

```
                maior = A[k];
```

```
            }
```

```
            else {
```

```
                if (A[k] > segMaior)
```

```
                    segMaior = A[k];
```

```
            }
```

```
        }
```

```
        k = k + 1;
```

```
    }
```

```
    // devolver resultado
```

```
    if (segMaior == maior) // não existe segundo maior negativo
```

```
        segMaior = 1;
```

```
    return segMaior;
```

```
}
```