UNIVERSIDADE DA BEIRA INTERIOR

Programação – LEI **1º Semestre**

Exame Época Normal **Resolução** **2023/2024**

1.

Escreva uma **expressão lógica em linguagem C** para a seguinte condição:

a) o valor da variável do tipo inteiro **N** é um número ímpar, mas não está entre 100 e 500

```
N % 2 == 1  &&  (N < 100  ||  N > 500)
N % 2 != 0  &&  (N < 100  ||  N > 500)
```

b) a soma dos valores das variáveis do tipo inteiro **N** e **M** é nula (zero), mas o valor de **N** é positivo

```
N + M == 0  &&  N >= 0
```

c) o valor da variável do tipo inteiro **N** é múltiplo de 5, mas não é divisor de 20

```
N % 5 == 0  &&  20 % N != 0
```

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

d) a variável do tipo inteiro **A** é decrementada em 10 valores

```
A = A – 10;
```

e) a variável do tipo inteiro **ARRED** recebe o valor arredondado da variável do tipo real **X**

```
ARRED = (int) (X + 0.5);
```

f) a variável do tipo inteiro **P** recebe o valor **10** se **N** for **par** ou **5** se **N** for **ímpar**

```
P = 10 - (N % 2) * 5;
```

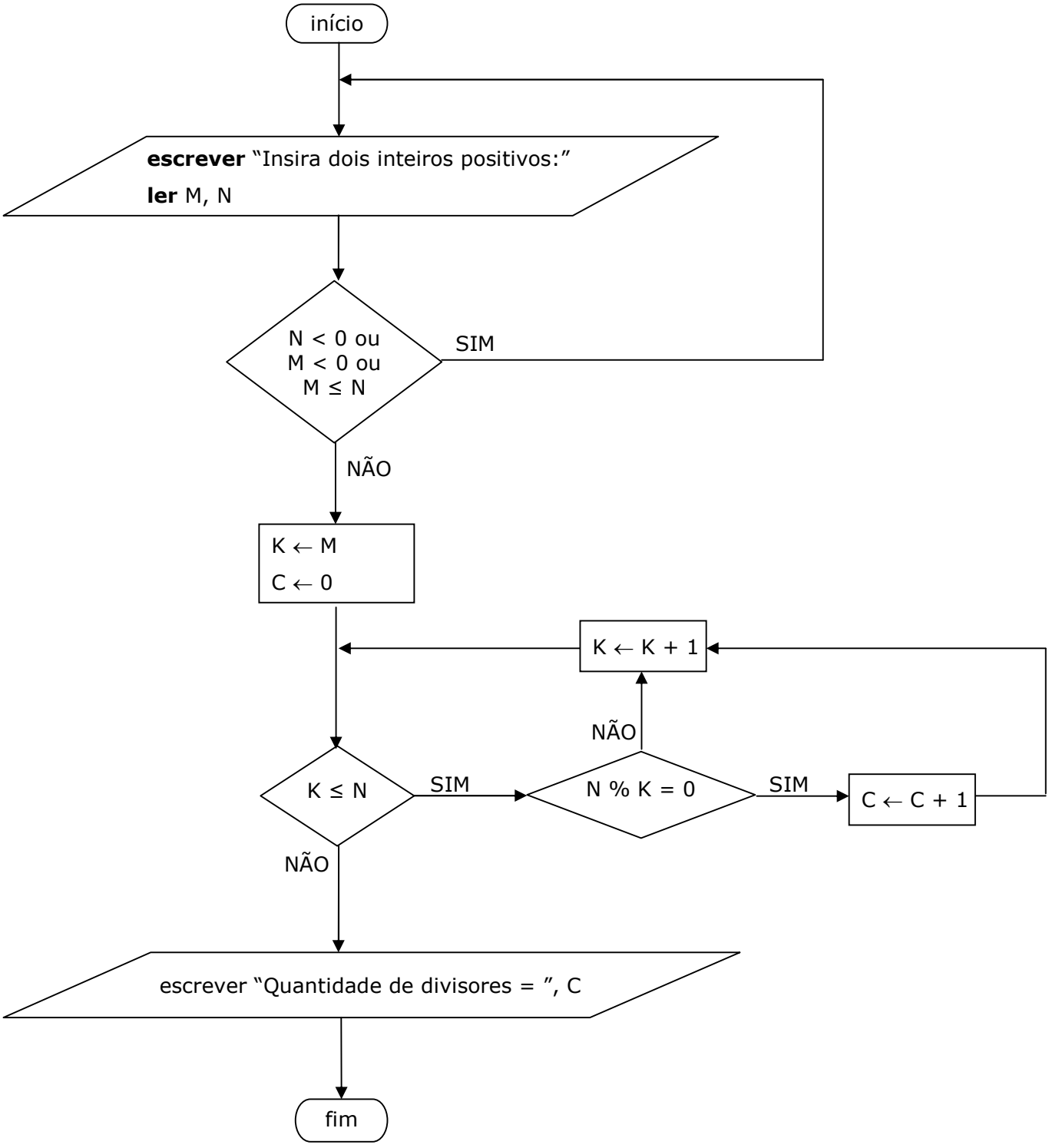
g) a variável do tipo inteiro **SOMA** recebe o valor da soma do algarismo das **centenas** da variável **A** com o algarismo das **unidades** da variável **B** (A e B são do tipo inteiro)

```
SOMA = (A % 100 / 10) + (B % 10);
```

2.

Usando um fluxograma, construa um algoritmo que realize as seguintes ações (pela ordem indicada):

1. peça ao utilizador que insira (leia) dois números inteiros positivos M e N, em que M < N
2. determine a quantidade de divisores de N entre M e N, incluindo eles próprios
3. mostre o resultado obtido (quantidade) com uma mensagem adequada.



3.

Construa um **programa em C** que realize as seguintes ações (pela ordem indicada):

1. peça ao utilizador que insira e leia dois números inteiros positivos não nulos **M** e **N**, em que M < N (M, N > 0 e M < N)
2. insira e leia **N** números inteiros positivos não nulos (> 0) e determine a **quantidade** destes números que são **divisores** de **M** e a **quantidade** destes números que são **múltiplos** de **M**
3. mostre os resultados obtidos no passo anterior (quantidade de divisores e quantidade de múltiplos) com uma mensagem adequada.

```
#include <stdio.h>
int main()
{
   int  M, N, A, k, quantDiv, quantMult;
   do{
      printf("Insira dois numeros inteiro positivos: ");
      scanf("%d%d", &M, &N);
   }while(M < 0  ||  N < 0  ||  M >= N);

   quantDiv = 0;
   quantMult = 0;
   k = 1;
   while (k <= N)
   {
      printf("Insira um inteiro positivo não nulo:");
      scanf("%d", &A);
      if (A > 0)
      {
         if (M % A == 0)
            quantDiv = quantDiv + 1;
         if (A % M == 0)
            quantMult = quantMult + 1;
         k = k + 1;
      }
   }

   printf("Quantidade de divisores: %d\n", quantDiv);
   printf("Quantidade de multiplos %d\n", quantMult);

}
```

4.

Implemente um **subprograma em C** que dado um array de 1 dimensão **X** com **N** números inteiros (X e N > 0 são parâmetros do subprograma), **devolva** como resultados a **quantidade** de elementos de **X** que são positivos (> 0) e a **média** desses elementos (dos positivos).

```
int  quantidadeMedia (int A[], int N, float *media)
{
   int k, soma, quant;
   soma = 0;
   quant = 0;
   for (k = 0; k < N; k++)
      if (A[k] > 0)
      {
         quant = quant + 1;
         soma = soma + A[k];
      }
   if (quant > 0)    // if (soma > 0)
      *media = (float) soma / quant;
   else
      *media = 0;
   return quant;
}
```

ou

```
float  quantidadeMedia (int A[], int N, int *quant)
{
   int k, soma;
   float  media;
   soma = 0;
   *quant = 0;
   for (k = 0; k < N; k++)
      if (A[k] > 0)
      {
         *quant = *quant + 1;
         soma = soma + A[k];
      }
   if (soma > 0)    //  if (*quant > 0)
      media = (float) soma / *quant;
   else
      media = 0;
   return media;
}
```

ou

...

5.

Considere as seguintes declarações de variáveis:

int *V, *W, **X;

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

(esquema de bloco de memória)

	...	
100500	110172	**V**
	...	
100800	110180	**W**
	...	
110160	10	
110164	20	
110168	30	
110172	40	
110176	50	
110180	60	
	...	
110500	100800	**X**
	...	

Considerando os valores que constam no esquema de um bloco de memória que se encontra ao lado, determine o valor de cada uma das seguintes expressões (**apresentar todos os cálculos efetuados**):

a) X + 2 = 100800 + 2 x sizeof(int*) = 100800 + 2 x 8 = **100816**

b) V[0] = **40**

c) W + 4 = 110180 + 4 x sizeof(int) = 110180 + 4 x 4 = **110196**

d) (*X)[0] = **60**

e) V + 2 = 110172 + 2 x sizeof(int) = 110172 + 2 x 4 = **110180**

f) *V + 3 = 40 + 3 = **43**

g) **X + 2 = 60 + 2 = **62**

h) X[0] + 2 = 110180 + 2 x sizeof(int) = 110180 + 2 x 4 = **110188**

i) *(W – 3) = *(110180 – 3 x sizeof(int)) = *(110180 – 3 x 4) = *(110168) = **30**

j) *(*X – 4) = *(110180 – 4 x sizeof(int)) = *(110180 – 4 x 4) = *(110164) = **20**

NOTA: se o valor da expressão não existir ou for desconhecido, então responder "**Indefinido**"

6.

Considere o ficheiro de texto "Inteiros.txt" apenas com números inteiros. Implemente um **programa em C** que realize as seguintes ações (pela ordem indicada):

1) **construa** um **array de 1 dimensão** com todos os números do ficheiro "Inteiros.txt", usando gestão de **memória dinâmica**,
2) **determine** a **quantidade** de números positivos (> 0) e a **quantidade** de números negativos (< 0) do array construído em 1),
3) **guarde** no ficheiro de texto "Saida.txt" os resultados obtidos no passo anterior.

NOTA: não é preciso verificar se há **ERRO** na **abertura dos ficheiros**, nem na **alocação de memória**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int  *A, tamA, num, k, neg, pos;
   FILE  *f, *g;
   // 1)
   f = fopen("Inteiros.txt", "r");
   tamA = 0;
   A = (int*) malloc(tamA * sizeof(int));
   while (fscanf (f, "%d", &num) == 1)
   {
      tamA = tamA + 1;
      A = (int*) realloc(A, tamA * sizeof(int));
      A[tamA-1] = num;
   }
   fclose(f);
   // 2)
   pos = 0;
   neg = 0;
   for (k = 0; k < tamA; k = k + 1)
   {
      if (A[k] > 0)
         pos = pos + 1;
      else
         if (A[k] < 0)
            neg = neg + 1;
   }
   // 3)
   g = fopen("Saida.txt", "w");
   fprintf(g, "Positivos: %d \n", pos);
   fprintf(g, "Negativos: %d \n", neg);
   fclose(g);
}
```

7.

Implemente um **subprograma em C** que dado um **array** de 1 dimensão **X** com **N** (N > 0) números inteiros (X e N são parâmetros do subprograma), **remova** do array todos os elementos repetidos e mantendo a ordem pela qual se encontram no array, tendo em conta que o array foi construído usando memória dinâmica.

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

Exemplo:

X = [3 -5 3 8 3 4 8 6 -2 -5] ==> X = [3 -5 8 4 6 -2]

```
int  removerRepetidos (int *X, int *N)
{
   int  i, j, k, existe;
   k = 0;
   while (k < *N)
   {
      // verificar se o valor de X[k] já existe nas posições anteriores a k
      existe = 0;
      i = 0;
      while (i < k && existe == 0)
      {
         if (X[i] == X[k])
            existe = 1;
         i = i + 1;
      }
      // se existe, remover X[k] do array, atualizar o tamanho e manter o valor de k
      // se não existe, avançar k uma posição no array
      if (existe == 1)
      {
         for (j = k; j < *N-1; j++)
            X[j] = X[j+1];
         *N = *N – 1;
      }
      else
         k = k + 1;
   }
   //  atualizar o array X, realocando memória apenas para o novo tamanho
   X = (int*) realloc(X, (*N) * sizeof(int));
}
```