UNIVERSIDADE DA BEIRA INTERIOR

Programação – LEI **1º Semestre**

Exame Época de Recurso **Resolução** **2023/2024**

1.

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

a) metade do valor da variável do tipo inteiro **N** é pelo menos igual ao resto da divisão de **N** por 5

```
N / 2 >= N % 5
```

b) o valor da variável do tipo real **X** não pertence ao intervalo [-5, 10]

```
X < -5  ||  X > 10
```

c) o valor da variável do tipo inteiro **N** é no mínimo 50 e no máximo 100, e não é múltiplo de 20

```
N >= 50  &&  N <= 100  &&  N % 20 != 0
N >= 50  &&  N <= 100  &&  N % 20 > 0
```

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

d) o valor da variável do tipo inteiro **A** é aumentado para o seu triplo

```
A = 3 * A;
```

e) a variável do tipo real **Z** recebe o valor da soma das partes inteiras das variáveis do tipo real **X** e **Y**

```
Z = (int) X + (int) Y;
```

f) a variável do tipo inteiro **P** recebe o valor **1** se **N** for par ou **-1** se **N** for ímpar

```
P = 1 - 2 * (N % 2);
```

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

```
SOMA = (A / 10 % 10) + (B / 1000 % 10);
SOMA = (A % 100 / 10) + (B % 10000 / 1000);
```

2.

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

1. peça ao utilizador que insira um número inteiro N entre 10 e 50, inclusivé
2. peça ao utilizador que insira N números reais e determine a **quantidade** de números positivos (> 0) inseridos e a **soma** dos números positivos (> 0) inseridos
3. mostre os resultados obtidos antes

3.

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

1. peça ao utilizador que insira um número inteiro N entre 1 e 50, inclusivé
2. peça ao utilizador para inserir N números inteiros não nulos (≠ 0) e, determine a **quantidade** de números positivos (> 0) **pares** e a **quantidade** de números positivos (> 0) **ímpares**
3. mostre os resultados obtidos antes (quantidade de pares e quantidade de ímpares)

```
#include <stdio.h>
  int main()
  {
     int  N, A, k, numPares, numImpares;
     do{
        printf("Insira um inteiro entre 1 e 50: ");
        scanf("%d", &N);
     }while(N < 1  ||  N > 50);

     numPares = 0;
     numImpares = 0;
     k = 1;
     while (k <= N)
     {
        printf("Insira um inteiro não nulo:");
        scanf("%d", &A);
        if (A > 0)
        {
           if (A % 2 == 0)
              numPares = numPares + 1;
           else
              numImpares = numImpares + 1;
           k = k + 1;
        }
     }

     printf("Quantidade de positivos pares: %d\n", numPares);
     printf("Quantidade de positivos impares %d\n", numImpares);

  }
```

4.

Implemente um **subprograma em C** que dado um array de 1 dimensão **X** com **N** (N > 0) números reais positivos (> 0) e um valor real positivo **K** (X, N e K são parâmetros do subprograma), **determine** e **devolva** como resultados a **quantidade** de elementos de X que são menores do que K e o **maior** elemento de **X** que é menor do que K.

NOTA: Se todos os elementos de X forem maiores do que K, então considerar maior = -1

```
int  quantidadeMenorK (float X[], int N, float K, float *maiorK)
{
   int  k, quant;
   *maiorK = -1;
   quant = 0;
   for (k = 0; k < N; k++)
   {
      if (A[k] < K)
      {
         quant = quant + 1;
         if (*maiorK < 0)  // ou if (*maiorK == -1)
            *maiorK = A[k];
         else
            if (A[k] > *maiorK)
               *maiorK = A[k];
      }
   }
   return quant;
}
```

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)

	...	
100350	110144	**V**
	...	
100725	100350	**W**
	...	
110132	12	
110136	15	
110140	16	
110144	13	
110148	14	
110152	17	
	...	
110585	110136	**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 = 110136 + 2 x sizeof(int) = 110136 + 8 = **110144**

b) V[0] = **13**

c) W + 4 = 110350 + 4 x sizeof(int*) = 110350 + 32 = **110382**

d) (*W)[1] = **14**

e) V + 2 = 110144 + 2 x sizeof(int) = 110144 + 8 = **110152**

f) *V + 3 = 13 + 3 = **16**

g) **W + 4 = 13 + 4 = **17**

h) W[0] + 4 = 110144 + 4 x sizeof(int) = 110144 + 16 = **110160**

i) X – 2 = 110136 – 2 x sizeof(int) = 110136 – 8 = **110128**

j) *(*W – 2) = *(110144 – 2 x sizeof(int)) = *(110136) = **15**

6.

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

1. usando gestão de **memória dinâmica**, **construa** dois **arrays de 1 dimensão**, um com todos os números positivos (> 0) e um outro com todos os números negativos (< 0) do ficheiro "reais.txt"
2. **guarde** no ficheiro de texto "saida.txt" todos os elementos do array de negativos construído em 1
3. **acrescente** ao ficheiro de texto "saida.txt" todos os elementos do array de positivos construído em 1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   float  *POS, *NEG, num;
   int  tamPOS, tamNEG;
   FILE  *f, *g;
   // 1)
   f = fopen("reais.txt", "r");
   tamPOS = 0;
   POS = (float*) malloc(tamPOS * sizeof(float));
   tamNEG = 0;
   NEG = (float*) malloc(tamNEG * sizeof(float));
   while (fscanf (f, "%f", &num) == 1){
      if (num > 0){
         tamPOS = tamPOS + 1;
         POS = (float*) realloc(POS, tamPOS * sizeof(float));
         POS[tamPOS-1] = num;
      }
      else
         if (num < 0){
            tamNEG = tamNEG + 1;
            NEG = (float*) realloc(NEG, tamNEG * sizeof(float));
            NEG[tamNEG-1] = num;
         }
   }
   fclose(f);
   // 2) e 3)
   g = fopen("saida.txt", "w");
   for (k = 0; k < tamNEG; k = k + 1)
      fprintf(g, "%f\n", NEG[k]);
   for (k = 0; k < tamPOS; k = k + 1)
      fprintf(g, "%f\n", POS[k]);
   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 que **não são repetidos** (são únicos ou só aparecem 1 vez) 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:

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

```
int  removerUnicos (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 o valor X[k] não existe antes de k, então verificar se existe nas posições depois de k
      if (existe == 0){
         i = k + 1;
         while (i < *N  &&  existe == 0){
            if (X[i] == X[k])
               existe = 1;
            i = i + 1;
         }
      }
      // se o valor de X[k] não existe repetido, então removê-lo do array; não avançar k
      if (existe ==0){
         for (j = k; j < *N-1; j++)
            X[j] = X[j+1];
         *N = *N – 1;
      }
      else
         k = k + 1;  // avançar k
   }
   //  atualizar o array X, realocando memória apenas para o novo tamanho
   X = (int*) realloc(X, (*N) * sizeof(int));
}
```