

Possível resolução

1.

Considere as seguintes declarações de variáveis:

int **V, *W, X[5];

	...	
100500	110176	W
	...	
100700	110164	X
	...	
110160	50	
110164	10	
110168	15	
110172	60	
110176	70	
110180	80	
	...	
110500	100700	V
	...	

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

Usando os valores que constam no esquema ao lado, indique, justificando com os cálculos efetuados, os valores de cada uma das seguintes expressões:

a) $X + 4 = 110164 + 4 \times \text{sizeof(int)} = 110164 + 16 = \mathbf{110180}$

b) $*X + 4 = 10 + 4 = \mathbf{14}$

c) $\&(W[0]) = W = \mathbf{110176}$

d) $*(W - 2) = *(110176 - 2 \times \text{sizeof(int)}) = *(110168) = \mathbf{15}$

e) $V - 2 = 100700 - 2 \times \text{sizeof(int*)} = 100700 - 16 = \mathbf{100684}$

f) $*(V + 2) = *(110164 + 2 \times \text{sizeof(int)}) = *(110172) = \mathbf{60}$

g) $**V + 2 = 10 + 2 = \mathbf{12}$

h) $W[2] = \mathbf{ERRO}$

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

2.

Implementar uma **função** em C que dados um array 1D **A** com **N** ($N > 0$) **números inteiros positivos não nulos** (**A** e **N** são parâmetros do subprograma), **determine e devolva** como resultados a **quantidade** de números **pares** e o **maior** número **par** do array **A** (se não existirem pares, maior = -1).

void pares (int *A, int N, int *quant, int *maior)

```
{
    int k;
    *quant = 0;
    *maior = -1;
    for(k = 0; k <= N-1; k++)
    {
        if(A[k] % 2 == 0){
            *quant = *quant + 1;
            if(A[k] > *maior)
                *maior = A[k];
        }
    }
}
```

3.

Considere um ficheiro de texto de nome "**dados.txt**" no qual cada uma das suas linhas contém **dois números inteiros positivos não nulos (> 0)**. Implemente um **programa** em C que

- usando **gestão dinâmica de memória, construa** um **array 1D** com os números do ficheiro "**dados.txt**" da seguinte forma: **leia** os dois números da cada uma das linhas do ficheiro e **acrescente** ao **array** o **maior** deles,
- **guarde** todos os **números pares do array** no ficheiro de texto "**pares.txt**" (um por linha).

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int num1, num2, *A, int N, k;
    FILE *fin, *fout;
    fin = fopen("dados.txt", "r");
    N = 0;
    A = (int*) malloc(N * sizeof(int));
    while(!feof(fin))
    {
        fscanf(fin, "%d%d", &num1, &num2);
        N = N + 1;
        A = (int*) realloc(A, N * sizeof(int));
        if(num1 > num2)
            A[N-1] = num1;
        else
            A[N-1] = num2;
    }
    fclose(fin);
    fout = fopen("pares.txt", "w");
    for(k = 0; k <= N-1; k++){
        if(A[k] % 2 == 0)
            fprintf(fout, "%d\n", A[k]);
    }
    fclose(fout);
}
```

4.

Implementar uma **função recursiva** em C que dados um **array 1D A** com **N** ($N \geq 1$) **números inteiros** (A e N são parâmetros da função), **determine o produto** dos números do array **A** que são positivos não nulos (> 0).

```
int produto (int *A, int N)
{
    int prod;
    // caso base/terminal
    if(N == 1)
        if(A[0] > 0)
            return A[0];
        else
            return 1;
    // case geral: N > 0
    prod = produto(A, N-1);
    if(A[N-1] > 0)
        return prod * A[N-1];
    else
        return prod;
}
```

ou

```
int produto (int *A, int N)
{
    int prod;
    // caso base/terminal
    if(N == 0)
        return 1;
    // case geral: N > 0
    prod = produto(A, N-1);
    if(A[N-1] > 0)
        return prod * A[N-1];
    else
        return prod;
}
```

5.

Implementar uma **função** em C que dados como parâmetros um **array 1D A** com **N** números **inteiros positivos todos diferentes** e um número inteiro **P**, **construa** e **devolva** um array 1D **B** com os **P** maiores números do array **A**. (**Nota**: otimizar o algoritmo, percorrendo os arrays o menos vezes possível)
Exemplo: A = [32 23 12 55 13 10 51 24] e P = 4 $\Rightarrow$ B = [55 51 32 24].

Ideia: colocar os P maiores elementos de A em B ordenados crescentemente: $B[0] > B[1] > \dots > B[P-1]$

int* P_MaioresElementos (int *A, int N, int P)

```
{
    int *B, k, kk, aux;
    B = (int*) malloc(P * sizeof(int));
    B[0] = A[0];
    for(k = 1; k < P; k++)
        B[k] = -1;
    for(k = 1; k < N; k++)
    {
        if(A[k] > B[P-1])
        {
            kk = P - 1;
            while(kk > 0 && B[kk-1] == -1)
                kk = kk - 1;
            // colocar A[k] na última posição do array B com valor < A[k] ou na primeira igual a "-1"
            B[kk] = A[k];
            // colocar B[kk] = A[k] na posição correta em B
            kk = kk - 1;
            while(kk >= 0)
            {
                if(B[kk] < B[kk+1]) {
                    aux = B[kk];
                    B[kk] = B[kk+1];
                    B[kk+1] = aux;
                    kk = kk - 1;
                }
                else // A[k] já está na posição correta
                    kk = -1;
            } // while
        } // if
    } // for
    return B;
}
```
