

# Ponteiros e Alocação de Memória (exercícios)

# Exercício 1 - Maior

- Faça uma função recursiva que receba um vetor de double e retorne qual é o maior valor neste vetor:

```
double encontra_maior(double v[], int n)
```

Exemplo de uso:

```
Digite n: 5
```

```
4
```

```
-2.9
```

```
3
```

```
9.5
```

```
-1
```

```
Maior: 9.5
```

# Exercício 2

- Faça uma função que recebe um vetor de números e retorne:
  - Média
  - Desvio padrão  $\longrightarrow \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$
  - Menor e maior número
- Todos esses valores devem ser retornados com parâmetros passados por referência.

# Exercício 3

- Escreva uma função que receba um **vetor inicial já ordenado (ordem crescente)** e um número (**novo\_num**);
- A função deve então retornar outro vetor (em **dest**) que contenha todos os elementos do vetor passado no parâmetro e o novo número inserido na posição correta (para manter o novo vetor ordenado).

Comprimento do vetor

`void adiciona_ordenado(double* dest, double *inicial, int n, double novo_num);`

- Por exemplo, para o vetor {4, 7, 8, 90, 100} e o número 95, o retorno da função será o vetor {4, 7, 8, 90, 95, 100};
- O programa deverá pré-alocar o vetor na chamada da função.

# Exercício 4

- Escreva uma função que receba uma string e retorne **outra string** com todos os caracteres invertidos (a string original não deve ser alterada):

```
char *inverte_texto(char *texto);
```

- Por exemplo, para “abcde” deve retornar “edcba”;
- Pode usar strlen para obter o comprimento da string (está no string.h).

# Exercício 5 (a)

- Crie um programa que leia um vetor de  $n$  números inteiros; Este vetor deve ser passado para uma função recursiva que irá contar quantos números são primos.
- **Não use colchetes!** Portanto, será preciso usar malloc e aritmética de ponteiros.

# Exercício 5 (b)

- Complemente o exercício anterior:
  - Após contar a quantidade de primos, crie um novo vetor, que conterá apenas os números primos do vetor original;
  - **Não use colchetes! Portanto, será preciso usar malloc e aritmética de ponteiros.**

# Exercício 6

- Implemente as funções sorteio, ganhou e imprime\_numero.

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

void sorteio(int **sorteado) {
    // Sorteia uma sequencia de 3 números: retorna um vetor
    // de tamanho 3 pelo parametro sorteado
}

int ganhou(int *n_usuario, int *sorteado) {
    // Compara se as duas sequencias de tres numeros sao
    // iguais
}

void imprime_numero(int *numeros) {
    // Imprima a sequencia de 3 numeros
}

int main() {
    int *n_user = malloc(sizeof(int) * 3);
    scanf("%d %d %d", &n_user[0], &n_user[1], &n_user[2]);

    int *num_sorteio;
    sorteio(&num_sorteio);

    imprime_numero(n_user);
    imprime_numero(num_sorteio);
    if (ganhou(num_sorteio, n_user))
        printf("Ganhou!!!\n");
    else
        printf("NAO Ganhou!\n");
    // Adiciona os frees aqui
    return 0;
}
```




# Exercício 7

- Faça uma função recursiva que calcule o valor de PI usando a série de Gregory:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

```
double calcula_pi(int n)
```



n é o número de elementos a considerar da série de Gregory

# Exercício 8

- *O professor ABC escreveu uma função, mas esqueceu para que ela servia... Você pode ajuda-lo a descobrir o que a função **misterio** faz?*
- Há risco dessa função recursiva executar indefinidamente?

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

```
int misterio(char t[], int c) {  
    if (t[c] == '\0')  
        return 0;  
    else  
        return 1 + misterio(t, c + 1);  
}
```

```
int main() {  
    char vetor[50];  
  
    ...  
  
    printf("%d\n", misterio(vetor, 0));  
  
    return 0;  
}
```