

# Curso de Programação em C

## Introdução à Programação

Lucas Maximiliano de Oliveira e Silva

Colégio Visconde de Porto Seguro

12 de setembro de 2022

# Sumário

## Revisão

I/O, Loop, Estruturas de Controle

## Matrizes

Exemplo 1

Exemplo 2

Ponteiros

## Funções

Estrutura de uma Função

Exemplo

Local VS. Global

Recursão

# Revisão

## Loops:

### I/O:

- ▶ Input → `scanf()`;
- ▶ Output → `printf()`;
- ▶ "Format-specifiers":
  - ▶ `%c`: char;
  - ▶ `%x`: hexadecimal;
  - ▶ `%i` : int (ou `%d`);
  - ▶ `%.nlf`: double.
- ▶ `\n`: Pular linha.

```
1 for(int i=0; i<2; i++) {  
2     printf("i = %d\n", i);  
3 }
```

```
1 int i = 2;  
2 while(i!=0) {  
3     printf("i = %d\n", i);  
4     i--;  
5 }
```

```
1 int i = 0;  
2 do {  
3     printf("i = %d\n", i);  
4     i--;  
5 } while(i>0);
```

**If-Else:** Decidir entre **2** estados;

**Switch:** Decidir entre **vários** estados.

# Matrizes

**Matriz:** Matrizes (*Array*) armazenam informação em blocos de memória.

- ▶ Tamanho  $n$  do Array é fixo;
- ▶ Index inicial é 0;
- ▶ Nome do Array == endereço do 1° elemento;
- ▶ Acessar elementos individuais com  $A[i]$ , sendo "i" o índice do elemento, ou *ponteiros*;

⋮
$A[0]$
$A[1]$
$A[2]$
⋮
$A[n-1]$
⋮

## Exemplo 1

## Exemplo 1

```
1 #include <stdio.h>
2 int main() {
3     //Declarar Array A[] de 5 elementos:
4     int A[5];
5     //Inicializar Array:
6     for(int k=0; k<5; k++) {
7         A[k] = k/2;
8         printf("A[%i] = %i\n", k, A[k]);
9     }
10    return 0;
11 }
```

## Output

```
A[0] = 0
A[1] = 0
A[2] = 1
A[3] = 1
A[4] = 2
```

## Exemplo 2

## Exemplo 2

Matrizes também podem ter mais de uma dimensão. Por exemplo tabelas (2 dimensões: horizontal e vertical.)

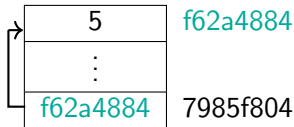
```
int A[m][n];    // m linhas e n colunas
```

A[0][0]	A[0][1]	A[0][2]	...	A[0][n-1]
A[1][0]	A[1][1]	A[1][2]	...	A[1][n-1]
⋮	⋮	⋮	⋱	⋮
A[m-1][0]	A[m-1][1]	A[m-1][2]	...	A[m-1][n-1]

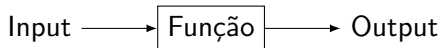
# Ponteiros

**Ponteiro:** Ponteiros são variáveis especiais, cuja informação que armazenam é o *endereço* de uma outra estrutura.

```
1 #include <stdio.h>
2 int main() {
3     int a = 5;
4     // Declarar um ponteiro:
5     int* ptr;
6     // Inicializar um ponteiro:
7     ptr = &a;
8     printf("Endereco a: %x", ptr);
9     printf("Valor a: %i", *ptr);
10    return 0;
11 }
```



# Funções



**Função:** Uma função é um trecho de código com um objetivo bastante específico.

- ▶ Objetivo: Compartimentar código;
  - ▶ Menos erros (localidade);
  - ▶ Mais legível;
- ▶ Declaração (funcoes.h) → Definição (funcoes.c) → Uso (main.c);



# Estrutura de uma Função

```
returnType functionName(argType arg1, argType arg2) {  
    // ...  
    return result  
}
```

- ▶ **returnType**: void, int, double, bool, etc.;
- ▶ **arg1, arg2**: Argumentos → input;
- ▶ **return**: Comando para retornar resultado → output;

## Exemplo

### main.c

```
1 #include <stdio.h>
2 //Tudo no mesmo diretorio
3 #include "funcoes.h"
4 int main() {
5     //Tamanho especificado
6     //pela lista:
7     int v[] = {0,1,7,3,9};
8     int w[] = {2,2,5,6,1};
9     int r = dotPrd(v,w,5);
10    printf("r = %i", r);
11    return 0;
12 }
```

### Output

r = 64

## Exemplo

## funcoes.h

```
1 #ifndef "funcoes.h"    //Header Guard
2 #define "funcoes.h"
3 #include "funcoes.c"   //Declarar:
4 int dotPrd(int v1[], int v2[], int n);
5 #endif
```

## funcoes.c

```
1 int dotPrd(int v1[], int v2[], int n) { //Definir:
2     int res = 0;
3     for(int i=0; i<n; i++) {
4         res += v1[i] * v2[i];
5     }
6     return res;
7 }
```

# Local VS. Global

**Global:** Variável acessível a todo programa;

**Local:** Variável acessível somente a partes do programa;

## Exemplo em Pseudo-código:

```
i = a //global
Foo1() {
    Foo2() {
        i = b //local
        print("print 1 : " i)
    }
    print("print 2 : " i)
}
```

## Output

```
print 1 : b
print 2 : a
```

# Recursão

**Recursão:** Quando uma função chama a si mesma. Cada "chamada" tem parâmetros diferentes e são conhecidas por *Stackframes*.

- ▶ Funções recursivas sempre possuem um caso não-recursivo, para evitar um ciclo infinito de repetições;
- ▶ Código bastante limpo, curto e elegante (mas nem sempre eficiente...);



## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

1  
2  
3  
4  
5  
6  
7  
8  
9

```
int fib(int n) {  
    if (n==0)  
        return 0;  
    if (n==1)  
        return 1;  
    else  
        return fib(n-1)  
            + fib(n-2);  
}
```

Stack para calcular fib(3):

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=3    **return** fib(2)+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=2    **return** fib(1)+fib(0)  
n=3    **return** fib(2)+fib(1)



## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=1	return 1
n=2	return fib(1)+fib(0)
n=3	return fib(2)+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=0	return 0
n=2	return 1+fib(0)
n=3	return fib(2)+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=2          **return** 1+0  
n=3    **return** fib(2)+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=3    return 1+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=1      return 1

n=3      return 1+fib(1)

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=3    return 1+1

## Exemplo

Calcular os n°s de Fibonacci: 0, 1, 1, **2**, 3, 5, 8, ...

```
1 int fib(int n) {  
2     if (n==0)  
3         return 0;  
4     if (n==1)  
5         return 1;  
6     else  
7         return fib(n-1)  
8             + fib(n-2);  
9 }
```

Stack para calcular fib(3):

n=3    return 1+1

Como melhorar? Caching →  
calcular fib(1) apenas x1