



# MCTA028-15: Programação Estruturada

## Aula 5: Recursão (Segunda Parte)

*Wagner Tanaka Botelho*

*wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com*

*Universidade Federal do ABC (UFABC)*

*Centro de Matemática, Computação e Cognição (CMCC)*

# Introdução

# Introdução

➤ Na **Linguagem C**:

➤ Uma **função** pode chamar **outras** funções;

➤ O **main()** pode chamar **qualquer** função, como por exemplo o **printf()** ou uma **função** definida pelo **programador**:

➤ A **função** implementada pelo **programador** também pode chamar **outras** funções.

➤ Vocês sabiam que uma **função** pode, **inclusive**, **chamar a si própria**?

➤ Isso é conhecido como **RECURSIVIDADE**, que é o processo de **REPETIR** alguma coisa de maneira **SIMILAR**.

Ex\_04.c x

```
1  #include<stdio.h>
2
3  int Soma(int a, int b){
4      int result=0;
5
6      result = a + b;
7
8      return result;
9  }
10
11 void main(){
12     int x=0, y=0, result=0;
13
14     printf("Digite dois numeros: \n");
15     scanf("%i", &x);
16     scanf("%i", &y);
17
18     result = Soma(x, y);
19
20     printf("O resultado da soma eh: %i", result);
21 }
```

## Exemplo no Mundo Real

+ Para melhor compreender o conceito, vamos utilizar um exemplo no **MUNDO REAL**:

**1**

O problema é:

Como esvaziar um vaso  
contendo TRÊS flores?



Etapa 1:

O vaso está vazio?



Etapa 2:

Não! Então, tira-se  
uma flor!



2

O problema é: Como esvaziar um vaso contendo DUAS flores?



Etapa 1: O vaso está vazio?



Etapa 2: Não! Então, tira-se uma flor!



3

O problema é:

Como esvaziar um vaso  
contendo UMA flor?



Etapa 1:

O vaso está vazio?



Etapa 2:

Não! Então, tira-se  
uma flor!



3

O problema é:

Como esvaziar um vaso  
contendo NENHUMA flor?



Etapa 1:

O vaso está vazio?



Etapa 2:

Sim! Então, FIM do  
processo!

Percebe-se que TODO o processo de  
REMOVER flores é MUITO REPETITIVO!!





A ideia das flores pode ser **generalizada**:

9/18

Para esvaziar um vaso contendo N flores, primeiro verifica-se se o vaso está vazio. Se o vaso não estiver vazio, tira-se uma flor (N-1).

```
*Ex_05.c X
1  #include<stdio.h>
2
3  void esvaziar(int qtdFlores){
4      if(qtdFlores>0){
5          qtdFlores--;
6          printf("Qtd. de flores no vaso: %i\n", qtdFlores);
7          esvaziar(qtdFlores);
8      }
9  }
10
11 void main(){
12     int qtd=3;
13
14     printf("Qtd. de flores no vaso: %i\n", qtd);
15     esvaziar(qtd);
16 }
```

Linha	qtd	qtdFlores
12	3	
14	{3}	
3		<del>3</del>
5		<del>2</del>
6		{2}
3		<del>1</del>
5		<del>1</del>
6		{1}
3		<del>1</del>
5		<del>0</del>
6		{0}
3		0

# Exemplo do Fatorial

Como calcular o fatorial de 3?



Para calcular o fatorial de 3, multiplica-se o número 3 pelo fatorial de 2 (definido como 2!).



Generalizando esse processo, tem-se que o fatorial de  $N$  é igual a  $N$  multiplicado pelo fatorial de  $(N - 1)$ , ou seja,  $N! = N * (N - 1)!$ .



Quando o processo termina?

Quando o número zero for atingido. Nesse caso, o valor do fatorial de 0 ( $0!$ ) é definido como igual a 1.

O fatorial de 3 é o produto de todos os números inteiros entre 1 e 3.

$$3! = 3 * 2 * 1$$

Aplicando a ideia da RECURSÃO!!

Etapa 1: O fatorial de 3 é definido em função do fatorial de 2;

Etapa 2: O fatorial de 2 é definido em função do fatorial de 1;

Etapa 3: O fatorial de 1 é definido em função do fatorial de 0;

Etapa 4: O fatorial de 0 é definido como igual a 1.

$$N! = N * (N - 1)!$$

$$3! = 3 * 2!$$

Portanto,  $3! = 6$ .

$$2 * 1!$$

$$2 * 1 = 2$$

Qual o fatorial de 1?

$$1 * 0!$$

$$1 * 1 = 1$$

Qual o fatorial de 0?

1

É 1!!!!

\*Ex\_07.c

1#include<stdio.h>

2

3double fatorial(double n){

4double r=0;

53! = 3 \* 2! = 6

6if (n==0){

7return 1;

8}1 \* 0! = 1

9else{

10r = n\*fatorial(n-1);

11return r;

12}

13}

14void main(){

15double num=3, res=0;

16

17res = fatorial(num);

18printf("Fatorial eh: %.1f \n", res);

19}

Linha	num	res	n	r
15	3	<del>0</del>		
3			<del>3</del>	
4				0
10				3*2!=?
3			<del>2</del>	
4				0
10				2*1!=?
3			<del>1</del>	
4				0
10				1*0!=?
3			0	
4				0
11				1
11				1
11				2
11				6
17		6		

Linha	num	res	n	r
18		{6}		

# Recursão

# Recursão

- As formas **recursivas** são consideradas “**mais enxutas**” e “**mais elegantes**” do que as formas **iterativas**;
- Muito cuidado na implementação de funções recursivas, pois **DOIS** critérios devem ficar bem estabelecidos:
  - **Critério de parada**;
  - **Parâmetro da chamada recursiva**.
- **Critério de parada**:
  - Determina quando a função deve **PARAR** de **CHAMAR** a **si mesma**. Se ela não existir, a função continuará executando até esgotar a memória do computador (*loop* infinito).;
  - No cálculo do **fatorial**, o **critério de parada** ocorre quando tenta-se calcular o **fatorial de zero**:  $0!=1$ .


# Recursão


## ➤ Parâmetro da chamada recursiva:

- O valor do **parâmetro passado** deve **SEMPRE** ser mudado, para que a recursão chegue a um **término**;
- Se o **parâmetro** for sempre o **mesmo**, a função continuara executando até **esgotar** a memória do computador (*loop* infinito);
- No **cálculo do fatorial**, a mudança no parâmetro da chamada recursiva ocorre quando o fatorial de N é definido em termos do fatorial de (N-1):  $N! = N * (N-1)!$ .



```
1  #include<stdio.h>
2
3  double fatorial(double n){
4      double r;
5
6      if(n==0){
7          return 1;
8      }
9      else{
10         r = n*fatorial(n-1);
11         return r;
12     }
13 }
14 void main(){
15     double num=3, res=0;
16
17     res = fatorial(num);
18     printf("Fatorial eh: %.1f \n", res);
19 }
```

 Critério de parada!

 Parâmetro do fatorial sempre muda!

# Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.