

### Universidade Federal do ABC (UFABC) Centro de Matemática Computação e Cognição (CMCC)

Recursividade PE-10 – v1.0

Prof. Paulo Joia Filho



- Introdução
- 2 Recursividade
- 3 Conclusões

- IntroduçãoObjetivos
- 2 Recursividade
- 3 Conclusões



# **Objetivos**

Os principais objetivos desta aula são:

- Apresentar o conceito de recursividade.
- Entender o funcionamento de uma função recursiva.
- Implementar alguns algoritmos de forma recursiva





## **Objetivos**

Os principais objetivos desta aula são:

- Apresentar o conceito de recursividade.
- Entender o funcionamento de uma função recursiva.
- Implementar alguns algoritmos de forma recursiva.





## **Objetivos**

Os principais objetivos desta aula são:

- Apresentar o conceito de recursividade.
- Entender o funcionamento de uma função recursiva.
- Implementar alguns algoritmos de forma recursiva.







- Introdução
- 2 Recursividade
  - Funções recursivas
  - Iteração x recursão
  - Exemplos adicionais
- 3 Conclusões



#### Recursão

- Conforme já estudado, em C, é comum uma função chamar outra.
- Em alguns casos, a outra função pode ser ela mesma.
- Uma função que chama a si mesma denomina-se função recursiva.
- Isto pode ocorrer de forma direta ou indireta.

#### Exemplo:

- ☐ f1 chama f1
- ☐ f1 chama f2 que chama f1



#### Recursão

- Alguns problemas são naturalmente recursivos.
- Um caso típico é o fatorial:

```
0! = 1! = 1
n! = n * (n-1)!
```

- Ao implementar uma função recursiva, é importante definir seu ponto de parada (caso base).
- No exemplo do fatorial, a recursão para quando atinge o valor 0 ou 1.



[fatorial\_recursivo.c]

#### Exemplo 1 (Fatorial Recursivo)

```
#include <stdio.h>
size_t fat(int); /* prototipo */
5 int main() {
     int n;
7
     printf("n = "); scanf("%d", &n);
     printf("%d! = %lu\n", n, fat(n));
     return 0:
11 }
size_t fat(int n) {
     if (n == 0 || n == 1) {
         return 1; /* ponto de parada */
15
     return n * fat(n - 1); /* codigo recursivo */
17
  }
```

## Função Recursiva

Uma função recursiva deve possuir duas partes básicas:

- Caso base
- Chamada recursiva: pelo menos uma chamada a si mesma.

#### Caso base

- É importante ter cuidado ao definir o caso base.
- Caso contrário, poderá ocorrer infinitas chamadas recursivas.
- Na verdade, o que ocorre nesse caso é o estouro da pilha de chamadas (stack overflow)
- O controle de chamadas a outros métodos é realizado por meio de uma pilha, conforme será explicado adiante.



```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```



```
int fatorial(int n =2 ) {
    if (n == 0)
        return 1;
    else
        return n *
                   int fatorial(int n =1 ) {
                        if (n == 0)
                            return 1;
                        else.
                            return n *
                                        int fatorial(int n =0 ) {
                                            if (n == 0)
                                                return 1;
                                            else.
                                                return n * fatorial(n - 1);
```

```
int fatorial(int n =2 ) {
    if (n == 0)
        return 1;
    else
        return n *
                   int fatorial(int n =1 ) {
                       if (n == 0)
                            return 1;
                       else.
                            return n *
                                        int fatorial(int n =0 ) {
                                            if (n == 0)
                                                return 1;
                                            else.
                                                return n * fatorial(n - 1);
```

```
int fatorial(int n =2 ) {
    if (n == 0)
        return 1;
    else
        return n * 1
}
```

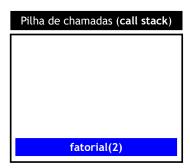
```
int fatorial(int n =2 ) {
    if (n == 0)
        return 1;
    else
        return n * 1
```

## Qual o valor de fatorial(2)?

```
int fatorial(int n) {
   if (n == 0)
      return 1;
   else
      return n * fatorial(n - 1);
```

2

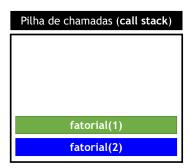
```
int fatorial(int n
   if (n == 0)
        return 1;
   else
        return n * fatorial(n - 1);
}
```





```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n
   if (n == 0)
        return 1;
   else
        return n * fatorial(n - 1);
}
```



```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n
   if (n == 0)
        return 1;
   else
        return n * fatorial(n - 1);
}
```

```
Pilha de chamadas (call stack)

fatorial(0)

fatorial(1)

fatorial(2)
```

```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n =1) {
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
Pilha de chamadas (call stack)

fatorial(0)
fatorial(1)
fatorial(2)
```

```
int fatorial(int n
    if (n == 0)
        return 1;
    else
        return n * fatorial(n - 1);
}
```

```
int fatorial(int n =1 ) {
    if (n == 0)
        return 1;
    else
        return n * 1;
}
```



```
int fatorial(int n =2 ) {
    if (n == 0)
        return 1;
    else
        return n * 1;
}
```

```
Pilha de chamadas (call stack)

fatorial(2)
```

Qual o valor de fatorial(2)?

Pilha de chamadas (call stack)

## Iteração × Recursão

- Em geral, uma função iterativa pode ser escrita de modo recursivo.
- A recíproca também é verdadeira: uma função recursiva pode ser escrita usando iteração.
- Muitas vezes a implementação recursiva é mais simples, porém, é computacionalmente menos eficiente que a versão iterativa.



### Série de Fibonacci

 A série de Fibonacci é definida da seguinte forma:

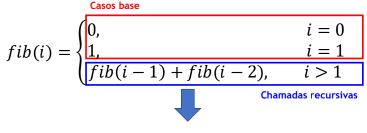
$$fib(i) = \begin{cases} 0, & i = 0 \\ 1, & i = 1 \\ fib(i-1) + fib(i-2), & i > 1 \end{cases}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



#### Série de Fibonacci

 A série de Fibonacci é definida da seguinte forma:



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



### Exemplo 2 (Fibonacci Recursivo)

```
#include <stdio.h>
size_t fib(int); /* prototipo */
5 int main() {
     int n;
7
     printf("n = "); scanf("%d", &n);
     printf("fib(%d) = %lu\n", n, fib(n));
     return 0:
11 }
size_t fib(int n) {
     if (n == 0 \mid \mid n == 1)  { /* ponto de parada */
         return n:
15
     return fib(n - 1) + fib(n - 2); /* codigo recursivo */
17
  }
```

#### Soma dos elementos de um vetor

[soma\_vetor.c]

```
Exemplo 3 (Versão Iterativa)
  #include <stdio.h>
  double sum_vector(double v[], int n) {
        int i:
       double sum = 0.0;
        for (i = 0; i < n; i++)
           sum += v[i];
       return sum;
  9 }
    int main() {
        double v[] = \{1, 2, 3, 4.5, 5.5\};
        int sz = sizeof(v) / sizeof(double);
  13
       printf("%.21f\n", sum_vector(v, sz));
  15
       return 0:
  17 }
```

### Soma dos elementos de um vetor

[soma\_vetor\_recursivo.c]

```
Exemplo 4 (Versão Recursiva)
  #include <stdio.h>
  double sum_vector_rec(double v[], int n) {
        if (n == 0)
           return v[0]:
       else
          return v[n] + sum_vector_rec(v, n-1);
  7
     }
    int main() {
       double v[] = \{1, 2, 3, 4.5, 5.5\};
        int sz = sizeof(v) / sizeof(double);
  13
       printf("%.21f\n", sum_vector_rec(v, sz));
       return 0;
  15
     }
```

- Introdução
- 2 Recursividade
- 3 Conclusões
  - Exercícios de aprendizagem
  - Considerações finais
  - Referências bibliográficas





#### Exercício 1 (Maior valor de um vetor)

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

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

#### Exemplo de uso:

```
Digite n: 5
4
-2.9
3
9.5
-1
Maior: 9.5
```

Exercício 2 (Comprimento de uma string)

 Faça uma função recursiva que receba uma string e retorne o comprimento dela.

#### Exemplo de uso:

Digite uma frase: Este eh um teste

Comprimento: 16



#### Exercício 3 (Decimal para hexadecimal)

 Faça uma função recursiva que receba um número decimal maior que zero e o imprima em hexadecimal.

```
void dec2hexa (int n)
```

#### Exemplo de uso:

```
Digite um numero: 333
14D
```



#### • Exemplo:

$$-333 \div 16 = 20 \text{ e resto } 13$$

$$-20 \div 16 = 1$$
 e resto 4

$$-1 \div 16 = 0$$
 e resto

• Portanto,  $(333)_{10} = (14D)_{16}$ 

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	В
12	С
13	D
14	E
15	F

#### Exercício 4 (Troca número primo)

 Faça uma função recursiva que troca todos os números primos de um vetor de inteiros por zero.

$$\{4, 6, 9, 23, 6, 11, 15\} \rightarrow \{4, 6, 9, 0, 6, 0, 15\}$$

#### Exemplo de uso:

```
Digite n: 6
4 8 3 11 20 29
4 8 0 0 1 0
```

# Considerações Finais

- Nesta aula foram apresentados os principais conceitos relacionados a:
  - Recursividade em C.

√ É importante rever os conceitos apresentados na aula e consultar a bibliografia sugerida sobre o assunto.



# Referências Bibliográficas I



Programação em C++: Algoritmos, Estruturas de Dados e Objetos.

McGraw-Hill, São Paulo.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002).

Algoritmos: Teoria e Prática.

Elsevier, Rio de Janeiro.



Estrutura de Dados e Algoritmos em C++.

Cengage Learning, São Paulo.



Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados.

Pearson Prentice Hall, São Paulo, 3 edition.



The Art of Computer Programming.

Addison-Wesley, Upper Saddle River, NJ, USA.

Pinheiro, F. d. A. C. (2012).

Elementos de Programação em C.

Bookman, Porto Alegre.

# Referências Bibliográficas II



Pisani, P. H. (2018).

Programação Estruturada - Notas de Aula.

Universidade Federal do ABC (UFABC).

Disponível em: <a href="http://professor.ufabc.edu.br/~paulo.pisani/">http://professor.ufabc.edu.br/~paulo.pisani/</a>.



Sedgewick, R. (1998).

Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, Searching.

Addison-Wesley, Boston, 3rd edition.



Szwarcfiter, J. L. e Markenzon, L. (1994).

Estruturas de Dados e Seus Algoritmos.

LTC, Rio de Janeiro.



Tenenbaum, A. A., Langsam, Y., e Augenstein, M. J. (1995).

Estruturas de Dados Usando C.

Makron Books, São Paulo.



Terra, R. (2014).

Linguagem C - Notas de Aula.

Universidade Federal de Lavras (UFLA).

Disponível em:

<http://professores.dcc.ufla.br/~terra/public\_files/2014\_apostila\_c\_ansi.pdf>. Acesso
em: 2018-09-16