



Recursão

Prof. Lilian Berton

São José dos Campos, 2019

Recursividade

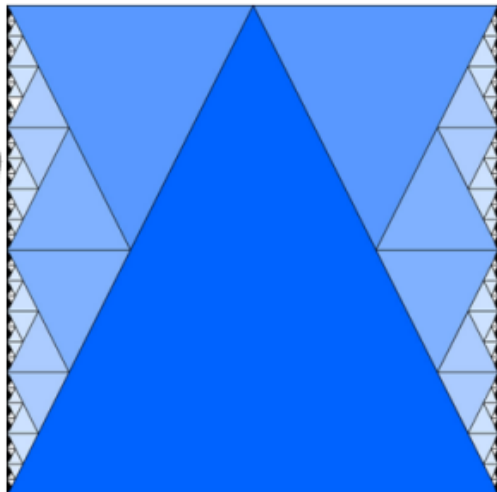
- **Recursividade** é um termo usado de maneira mais geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora mostrado.
- Por exemplo, segue uma definição recursiva da ancestralidade de uma pessoa:
 - Os pais de uma pessoa são seus antepassados (*caso base*);
 - Os pais de qualquer antepassado são também antepassados da pessoa em consideração (*passo recursivo*).



Exemplos de recursão



Fractal espiral



Fractal de triângulos



Fractal de Fern





Recursividade em C

- Na linguagem C, assim como em muitas outras linguagens de programação, uma função pode chamar a si própria.
- Uma função assim é chamada **recursiva**.
- Devemos tomar cuidado ao se fazer funções recursivas. A primeira coisa a se providenciar é um **critério de parada**.
- Este vai determinar **quando a função deverá parar de chamar a si mesma**. Isto impede que a função se chame infinitas vezes.

Recursividade em C

- Estratégia para a definição recursiva de uma função:
 1. **Dividir o problema em problemas menores do mesmo tipo**
 2. **Resolver os problemas menores (dividindo-os em problemas ainda menores, se necessário)**
 3. **Combinar as soluções dos problemas menores para formar a solução final**
- Ao dividir o problema sucessivamente em problemas menores eventualmente os casos simples são alcançados: não podem ser mais divididos ou suas soluções são definidas explicitamente.

Recursividade em C

- Uma definição de função recursiva é dividida em duas partes:
 1. Há um ou mais casos base que dizem o que fazer em situações simples, onde não é necessária nenhuma recursão. Nestes casos a resposta pode ser dada de imediato, sem chamar recursivamente a função sendo definida. Isso garante que a recursão eventualmente possa parar.
 2. Há um ou mais casos recursivos que são mais gerais, e definem a função em termos de uma chamada mais simples a si mesma.
- fatorial n
- $n == 0 \rightarrow 1$  **Caso base**
- $n > 0 \rightarrow \text{fatorial}(n-1) * n$  **Caso recursivo**

Exemplo fatorial recursivo

```
fatorial 6
```

```
↪ fatorial 5 * 6
```

```
↪ (fatorial 4 * 5) * 6
```

```
↪ ((fatorial 3 * 4) * 5) * 6
```

```
↪ (((fatorial 2 * 3) * 4) * 5) * 6
```

```
↪ ((((fatorial 1 * 2) * 3) * 4) * 5) * 6
```

```
↪ ((((((fatorial 0 * 1) * 2) * 3) * 4) * 5) * 6
```

```
↪ ((((((1 * 1) * 2) * 3) * 4) * 5) * 6
```

```
↪ (((((1 * 2) * 3) * 4) * 5) * 6
```

```
↪ (((2 * 3) * 4) * 5) * 6
```

```
↪ ((6 * 4) * 5) * 6
```

```
↪ (24 * 5) * 6
```

```
↪ 120 * 6
```

```
↪ 720
```

Exemplo fatorial recursivo

```
#include <stdio.h>
int fat(int n);

int main() {
    int n;
    printf("\n\nDigite um valor para n:");
    scanf("%d", &n);
    printf("\nO fatorial de %d eh %d",n,fat(n));
    return 0;
}

int fat (int n){
    int fatorial = 1;
    for (int i = n; i > 0; i--) {
        fatorial = fatorial * i;
    }
    return fatorial;
}
```

Não usa recursão

```
#include <stdio.h>
int fat(int n);

int main() {
    int n;
    printf("\n\nDigite um valor para n:");
    scanf("%d", &n);
    printf("\nO fatorial de %d eh %d",n,fat(n));
    return 0;
}

int fat (int n){
    if(n>0)
        return n*fat(n-1);
    else return 1;
}
```

Usa recursão

Exemplo fatorial recursivo

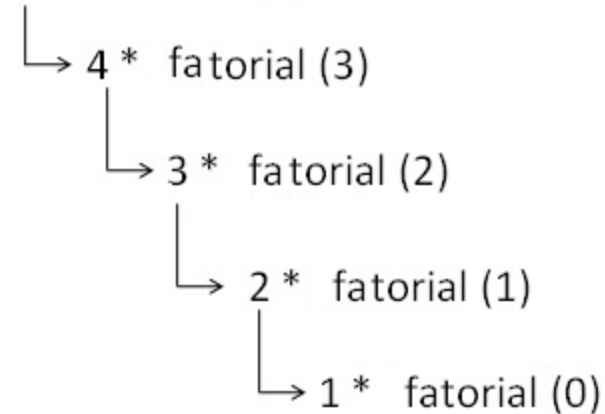
```
#include <stdio.h>
int fat(int n);

int main() {
    int n;
    printf("\n\nDigite um valor para n:");
    scanf("%d", &n);
    printf("\nO fatorial de %d eh %d",n,fat(n));
    return 0;
}

int fat (int n){
    if(n>0)
        return n*fat(n-1);
    else return 1;
}
```

n=0 é o critério de parada para esta função.

fatorial (5) = 5 * fatorial (4)



	120
5 *	24
4 *	6
3 *	2
2 *	1
1 *	1

**O compilador
gera uma pilha
com as chamadas
recursivas**

Exemplo multiplicação recursiva

- A multiplicação de inteiros está disponível na biblioteca como uma operação primitiva por questões de eficiência.
- Porém ela pode ser definida usando recursividade em um de seus argumentos:
- $\text{Mul}(\text{int } m, \text{int } n)$
- $n == 0 = 0$ **Quando o multiplicador é zero, o produto também é zero.
Este é o caso base**
- $n > 0 = m + \text{mul}(m, (n-1))$ **$m * n = m + m * (n-1)$, sendo $n > 0$.
Este é um caso recursivo**
- $n < 0 = -(\text{mul}(m, (-n)))$ **$m * n = -(m * (-n))$, sendo $n < 0$.
Este é outro caso recursivo**

Exemplo multiplicação recursiva

```
mul 7 (-3)
~> - (mul 7 3)
~> - (7 + mul 7 2)
~> - (7 + (7 + mul 7 1))
~> - (7 + (7 + (7 + mul 7 0)))
~> - (7 + (7 + (7 + 0)))
~> - (7 + (7 + 7))
~> - (7 + 14)
~> - 21
~> -21
```

Exemplo multiplicação recursiva

```
#include <stdio.h>
int mul(int n, int m);

int main() {
    int n, m;
    printf("\n\nDigite dois valores para fazer a multiplicacao:");
    scanf("%d %d", &n, &m);
    printf("\nA multiplicacao de %d*%d eh %d",n,m,mul(n,m));
    return 0;
}

int mul(int n, int m){
    if (n > 0) {
        return m+(mul(m, n-1));
    } else if (n < 0) {
        return -(mul(m,(-n)));
    } else {
        return 0;
    }
}
```

Para o caso de num negativo,
Entra no if $n < 0$ e chama a função recursiva transformando o num neg. para positivo.
Na próxima vez entraria em $n > 0$.

Considerações

- Há certos algoritmos que são mais eficientes quando feitos de maneira recursiva.
- Mas a recursividade é algo a ser evitado sempre que possível, pois, se usada incorretamente, **tende a consumir muita memória e ser lenta.**
- Lembre-se que memória é consumida cada vez que o computador faz uma chamada a uma função.
- Uma função recursiva sempre deve ter um **caso de parada senão entrará num *loop* infinito.**

Exercícios

1. Crie um programa que use recursão para calcular a potência de 2 para números naturais.

```
pot2 4
~> 2 * pot2 3
~> 2 * (2 * pot2 2)
~> 2 * (2 * (2 * pot2 1))
~> 2 * (2 * (2 * (2 * pot2 0)))
~> 2 * (2 * (2 * (2 * 1)))
~> 2 * (2 * (2 * 2))
~> 2 * (2 * 4)
~> 2 * 8
~> 16
```

2. Crie um programa em C que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário introduziu ou seja: $1 + 2 + 3 + \dots + n$. Utilize recursividade.
3. Faça um programa que calcule divisão usando subtrações sucessivas e recursão.
i-nt div (int a, int b);

Recursão infinita

- The boss calls his secretary & says: "Get ready for d weekend, We r going on a business trip."

The secretary calls husband & says: "Me & my boss r going on a business trip for 2 days so take care of urself"

The husband calls his girlfriend & says: "My wife is going on a business trip come home we can have fun"

The girlfriend calls the boy to whom she gives tuition: "No tuition this weekend."

The boy calls his father: "Dad, at last we can spend this weekend together."

Dad (The boss) calls his secretary & says: "Business trip is cancelled. I'm going to spend weekend with my son"

The secretary calls husband: "I won't be going"

The husband calls his girlfriend: "I am sorry My wife is not going "

The girlfriend calls boy: "You have tuition"

Boy calls his father & says: "Sorry Dad, I've classes"

The Dad calls his secretary.....(infinite recursion)