Aula relâmpago

...

Compreendendo o comportamento das pilhas nas funções recursivas

Lucas Carvalho Corrêa
Formando em Sistemas de Informação na PUC - Minas

Definição de função recursiva

“Uma função que é dita recursiva é aquela que invoca ela mesma.”

http://www.cprogressivo.net/2013/03/O-que-sao-e-como-usar-funcoes-recursivas-em-linguagem-C.html

Vamos ver na pratica

Começando pelo simples:

Escreva uma função recursiva, que receba um numero inteiro unsigned e em seguida some a partir deste numero todos os números anteriores a ele.

Dado 5: 5+4+3+2+1 = 15

Vamos a solução

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1) + num;
    }
}
```

Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1) +
num;
    }
}
```

num = 1

num = 2

num = 3

num = 4

num = 5

Chamada do metodo no Main

Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1) +
num;
    }
}
```

num = 1

num = 2

num = 3

num = 4

num = 5

Chamada do metodo no Main

Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1) +
num;
    }
}
```

num = 1

num = 2

num = 3

num = 4

num = 5

Chamada do metodo no Main

Entendendo as pilhas

```
        static int somarNumerosAnteriores(int num)
        {
            if (num <= 1) {
                return 1;
            }
            else
            {
                return somarNumerosAnteriores(num - 1) +
num;
            }
        }
```

num = 1

num = 2

num = 3

num = 4

num = 5

Chamada do metodo no Main

Entendendo as pilhas

```
        static int somarNumerosAnteriores(int num)
        {
            if (num <= 1) {
                return 1;
            }
            else
            {
                return somarNumerosAnteriores(num - 1) +
num;
            }
        }
```

num = 1

num = 2

num = 3

num = 4

num = 5

Chamada do metodo no Main

Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1) +
num;
    }
}
```

num = 1

num = 2

num = 3

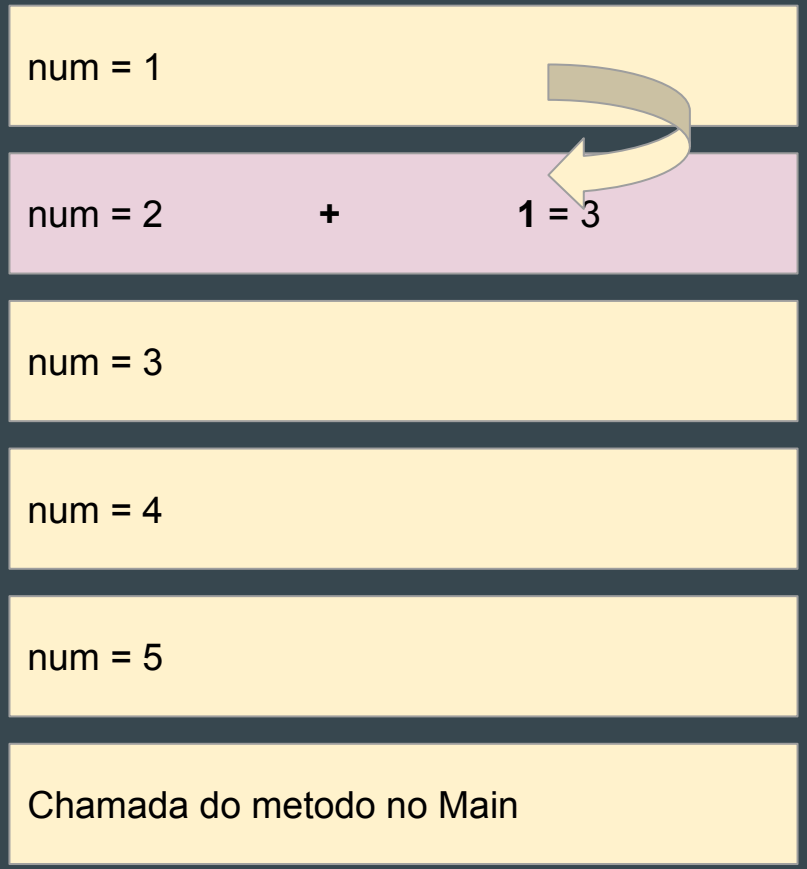
num = 4

num = 5

Chamada do metodo no Main

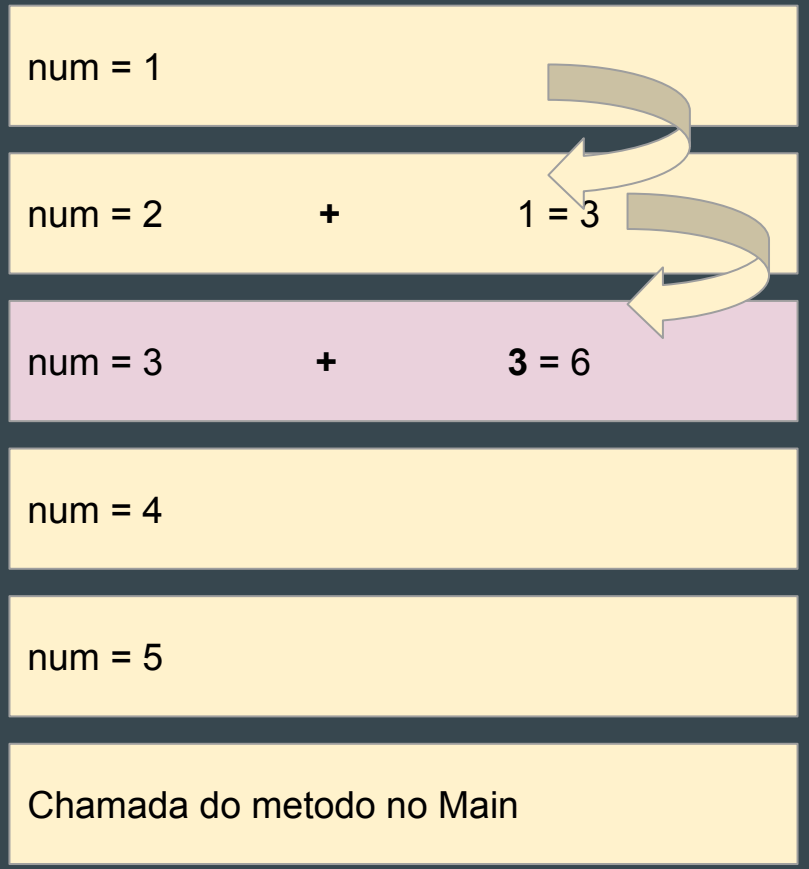
Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1)
+ num;
    }
}
```



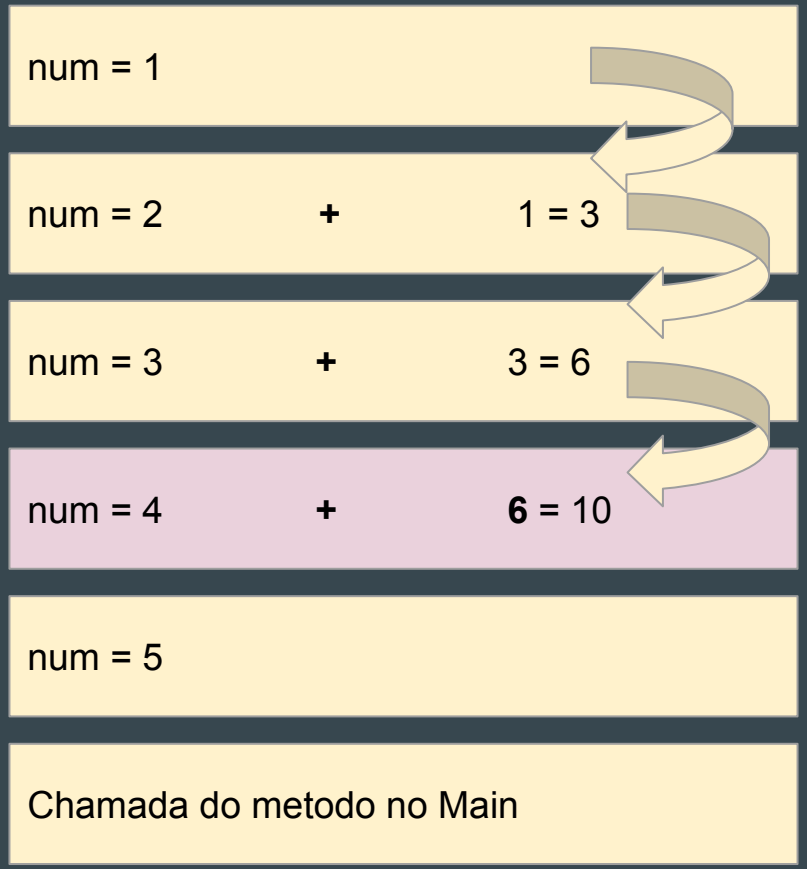
Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1)
+ num;
    }
}
```



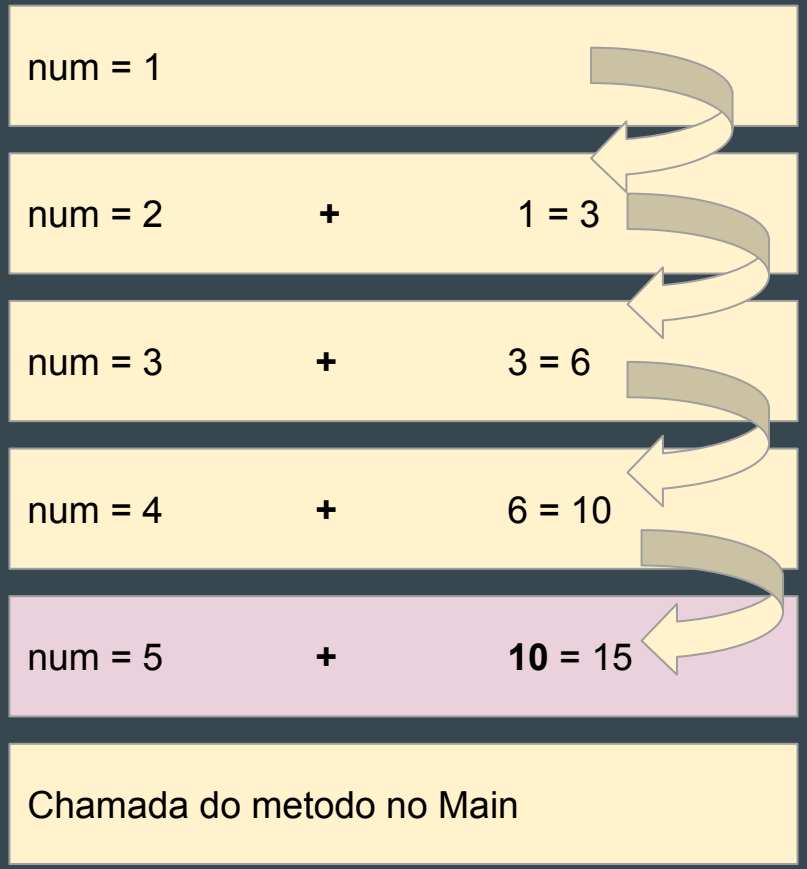
Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1)
+ num;
    }
}
```



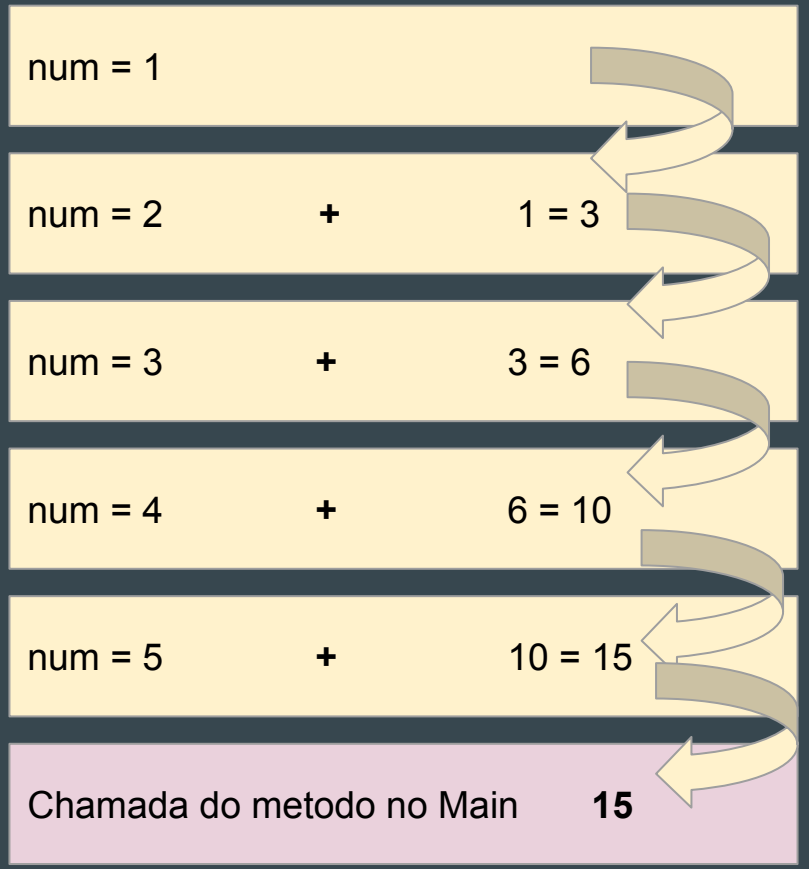
Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1)
+ num;
    }
}
```



Entendendo as pilhas

```
static int somarNumerosAnteriores(int num)
{
    if (num <= 1) {
        return 1;
    }
    else
    {
        return somarNumerosAnteriores(num - 1)
+ num;
    }
}
```



Problema mais elaborado

Escreva a função recursiva que determina o menor elemento de um vetor de inteiros.

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

Main ultiPos 3 , [3, 1, 2, 9]

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos 3 , [3, 1, 2, 9]

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **2** | aux: **5** | menor: **?**

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos 3 , [3, 1, 5, 9]

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **1** | aux:1 | menor: **?**

ultiPos: **2** | aux: **5** | menor: **?**

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos 3 , [3, 1, 5, 9]

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **0** | return **3**

ultiPos: **1** | aux:1 | menor: **?**

ultiPos: **2** | aux: **5** | menor: **?**

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos: **3** , **[3, 1, 5, 9]**

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **0** | return **3**

ultiPos: **1** | aux:**1** | menor: **3**

ultiPos: **2** | aux: **5** | menor: **?**

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos: **3** , **[3, 1, 5, 9]**

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **0** | return **3**

ultiPos: **1** | aux:**1** | menor: **3**

ultiPos: **2** | aux: **5** | menor: **1**

ultiPos: **3** | aux: **9** | menor: **?**

Main ultiPos: **3** , **[3, 1, 5, 9]**

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```

ultiPos: **0** | return **3**

ultiPos: **1** | aux:**1** | menor: **3**

ultiPos: **2** | aux: **5** | menor: **1**

ultiPos: **3** | aux: **9** | menor: **1**

Main ultiPos: **3** , **[3, 1, 5, 9]**

Entendendo as pilhas

```
static int encontrarMenorValor(int ultiPos, int[] vetor)
{
    if(ultiPos == 0)
    {
        return vetor[ultiPos];
    }
    else
    {
        int aux = vetor[ultiPos];
        int menor = encontrarMenorValor(ultiPos - 1, vetor);
        if(aux < menor)
        {
            menor = aux;
        }
        return menor;
    }
}
```