

Recursividade

“Uma das etapas da solução de um problema pode ser simplesmente uma versão do mesmo problema que você está tentando resolver pela primeira vez”.

Recursividade

Existem casos em que uma função chama a si própria. Diz-se então que a função é recursiva.

Exemplo 1

```
void contRegressiva(int n){  
    if (n == 0)  
        printf("\nFogo!");  
    else {  
        printf("\n%i",n);  
        contRegressiva(n-1);  
    }  
}
```

contRegressiva(5)

Recursividade

Toda vez que uma função é chamada, cria-se um novo quadro (*frame*) para a função na memória, que contém todas as variáveis locais e parâmetros da função.

Memória		
PP	n	tela
contRegressiva()	3	3
contRegressiva()	2	2
contRegressiva()	1	1
contRegressiva()	0	Fogo!

A parte mais em baixo na pilha, onde $n=0$, é chamada de **caso base**.

Exemplo 2

```
void pula(int n){  
    //Salta n linhas.  
    if (n > 0){  
        printf("\n");  
        pula(n-1); }  
}
```

Exemplo 2

```
void pula(int n){  
    //Salta n linhas.  
    if (n > 0){  
        printf("\n");  
        pula(n-1); }  
}
```

Programa Principal

```
int lin = 3;  
printf("Pulando...");  
pula(lin);  
printf("Pulei %i linhas.", lin);
```

Recursividade permite escrever algoritmos para solução de **problemas de natureza recursiva** de forma mais clara e concisa.

Exemplo 3

O fatorial de um número N pode ser definido por:

$$N! = \begin{cases} N.(N - 1)!, & \text{se } N \geq 1 \\ 1, & \text{se } N = 0 \end{cases}$$

Escreva uma função recursiva que traduza esta definição.

Exemplo 3

O fatorial de um número N pode ser definido por:

$$N! = \begin{cases} N.(N-1)!, & \text{se } N \geq 1 \\ 1, & \text{se } N = 0 \end{cases}$$

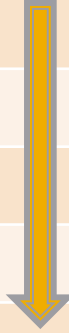
```
float f(int n){  
    //Calcula Fatorial.  
    float fat;  
    if (n == 0)  
        fat = 1;  
    else  
        fat = n*f(n-1);  
    return fat;  
}
```

Exemplo 3

```
float f(int n){  
    //Calcula Fatorial.  
    float fat;  
    if (n == 0)  
        fat = 1;  
    else  
        fat = n*f(n-1);  
    return fat;  
}
```

Teste de Mesa

n	fat
4	$4 * f(3)$
3	$3 * f(2)$
2	$2 * f(1)$
1	$1 * f(0)$
0	1




Exemplo 3

```
float f(int n){  
    //Calcula Fatorial.  
    float fat;  
    if (n == 0)  
        fat = 1;  
    else  
        fat = n*f(n-1);  
    return fat;  
}
```

Teste de Mesa

n	fat	f(n)
4	$4 * f(3)$	24
3	$3 * f(2)$	6
2	$2 * f(1)$	2
1	$1 * f(0)$	1
0	1	1



Programa Recursivo: vantagens

- *Em geral é mais elegante e menor que a sua versão iterativa;*

Programa Recursivo: vantagens

- É mais elegante e menor que a sua versão iterativa;
- *Exibe com maior clareza o processo utilizado, quando o problema ou dados são naturalmente definidos através da recorrência;*

Programa Recursivo: vantagens

- É mais elegante e menor que a sua versão iterativa;
- Exibe com maior clareza o processo utilizado, quando o problema ou dados são naturalmente definidos através da recorrência;
- ***Deve tornar o problema mais simples.***

Programa Recursivo: desvantagens

- É mais lento do que a versão iterativa;
- As diversas chamadas recursivas ocorrem em **instâncias diferentes** como se houvessem várias funções iguais sendo chamadas;

Programa Recursivo: desvantagens

- É mais lento do que a versão iterativa;
- As diversas chamadas recursivas ocorrem em instâncias diferentes como se houvessem várias funções iguais sendo chamadas;
- Cada instância conserva separadamente suas próprias **variáveis locais**, ou seja, as variáveis utilizadas em uma instância são diferentes daquelas utilizadas nas outras instâncias;

Programa Recursivo: desvantagens

- É mais lento do que a versão iterativa;
- As diversas chamadas recursivas ocorrem em instâncias diferentes como se houvessem várias funções iguais sendo chamadas;
- Cada instância conserva separadamente suas próprias variáveis locais, ou seja, as variáveis utilizadas em uma instância são diferentes daquelas utilizadas nas outras instâncias;
- ***Exige mais espaço de memória.***

Exemplo 4

```
int soma(int v[], int i){  
    // Soma recursiva de um vetor.  
    int s;  
    if (i == 0)  
        s = v[0];  
    else  
        s = v[i] + soma(v, i-1);  
    return s;  
}
```


Exemplo 4

```
int soma(int v[], int i){  
    // Soma recursiva de um vetor.  
    int s;  
    if (i == 0)  
        s = v[0];  
    else  
        s = v[i] + soma(v, i-1);  
    return s;  
}
```

```
int main(){  
    int n = 5, vet[n];  
    for (int i=0; i<n; ++i){  
        printf("\nnum[%d]: ", i);  
        scanf("%i", &vet[i]);  
    }  
    printf("\n\tSoma = %i", soma(vet, n-1));  
}
```

Exemplo 5

A série de Fibonacci pode ser definida por:

$$\text{Fib}(N) = \begin{cases} \text{Fib}(N - 1) + \text{Fib}(N - 2), & \text{se } N > 2 \\ 1, & \text{se } N = 1 \text{ ou } N = 2 \end{cases}$$

Criar uma função recursiva em Python que gere o termo de ordem N da série de Fibonacci.

Exemplo 5


```
int fibRec(int n){  
    //Gera o termo de ordem N da série de Fibonacci.  
    int fib;  
    if (n==1 || n==2)  
        fib = 1;  
    else  
        fib = fibRec(n - 1) + fibRec(n - 2);  
    return fib;  
}
```

Exemplo 5

```
int fibRec(int n){  
    //Gera o termo de ordem N da série de Fibonacci.  
    int fib;  
    if (n==1 || n==2)  
        fib = 1;  
    else  
        fib = fibRec(n - 1) + fibRec(n - 2);  
    return fib;  
}
```

Teste de mesa fibRec(4)

n	fib
4	fibRec(3) + fibRec (2)
3	fibRec(2) + fibRec (1)
2	1
1	1



Exemplo 5

```
int fibRec(int n){  
    //Gera o termo de ordem N da série de Fibonacci.  
    int fib;  
    if (n==1 || n==2)  
        fib = 1;  
    else  
        fib = fibRec(n - 1) + fibRec(n - 2);  
    return fib;  
}
```

Teste de mesa fibRec(4)

n	fib	fibRec(n)
4	fibRec(3) + fibRec (2)	3
3	fibRec(2) + fibRec (1)	2
2	1	1
1	1	1

Exemplo 5

Criar uma função NÃO RECURSIVA que gere o termo de ordem N da série de Fibonacci.

Exemplo 5

```
int fibNaoRec(int n){  
    int f1, f2, fib;  
    if (n==1 || n==2)  
        return 1;  
    f1 = f2 = 1;  
    for (int k=0; k<(n-2); k++){  
        fib = f1 + f2;  
        f1 = f2;  
        f2 = fib;  
    }  
    return fib;  
}
```

Exemplo 5

Criar uma função NÃO RECURSIVA que gere o termo de ordem N da série de Fibonacci.

```
int fibNaoRec(int n){
    int f1, f2, fib;
    if (n==1 || n==2)
        return 1;
    f1 = f2 = 1;
    for (int k=0; k<(n-2); k++){
        fib = f1 + f2;
        f1 = f2;
        f2 = fib;
    }
    return fib;
}
```

Teste de mesa de fibNaoRec(6)

f1	f2	k	fib
1	1	0:3	2
1	2	1	3
2	3	2	5
3	5	3	8
5	8		

Exemplo 5

```
int fiblter(int n) { //outra versão
    //Não Recursivo.
    int i = 1, fib = 0;
    for (int k=0; k<n; k++){
        fib += i;
        i = fib - i;
    }
    return fib;
}
```

Exemplo 5

```
int fiblter(int n) { //outra versão
    //Não Recursivo.
    int i = 1, fib = 0;
    for (int k=0; k<n; k++){
        fib += i;
        i = fib - i;
    }
    return fib;
}
```

Teste de mesa fiblter(5)

i	K=0:4	fib
1	0	0
0	1	1
1	2	1
1	3	2
2	4	3
3		5

Comparação de tempo

Recursivo x Iterativo

n	10	20	30	50	100
fibIter(n)	... ms	... ms	... ms	... ms	... ms
fibRec(n)	... ms	16 ms	2 seg	8 hs	16.000.000 anos

Comparação de tempo

Recursivo x Iterativo

n	10	20	30	50	100
fibIter(n)	... ms	... ms	... ms	... ms	... ms
fibRec(n)	... ms	16 ms	2 seg	8 hs	16.000.000 anos

Portanto, Fibonacci é um exemplo onde não se deve utilizar recursividade.

EXERCÍCIOS

- 1) Escrever uma função para calcular o Máximo Divisor Comum (MDC) de dois números dados como parâmetros. Sabe-se que para MDC de dois números x e y temos:

$$\text{MDC}(x, y) = \text{MDC}(x - y, y), \quad \text{se } x > y$$

$$\text{MDC}(x, y) = \text{MDC}(y, x), \quad \text{se } x < y$$

$$\text{MDC}(x, y) = x, \quad \text{se } x = y$$

EXERCÍCIOS

Sabe-se que para MDC de dois números x e y temos:

$$\text{MDC}(x, y) = \text{MDC}(x - y, y), \quad \text{se } x > y$$

$$\text{MDC}(x, y) = \text{MDC}(y, x), \quad \text{se } x < y$$

$$\text{MDC}(x, y) = x, \quad \text{se } x = y$$

```
int mdc(int x, int y){  
    //Calcula o Máximo Divisor Comum (MDC).  
    int z;  
    if (x == y)  
        return x;  
    if (x > y)  
        z = mdc(x - y, y);  
    else // x < y  
        z = mdc(y, x);  
    return z;  
}
```

```
mdc(4,2)  
mdc(2,4)  
mdc(3,2)
```

EXERCÍCIOS

2) Dado um número inteiro não negativo, crie uma função recursiva que escreva o mesmo na tela em vertical.

Entrada: 1234

Saída:

1

2

3

4

EXERCÍCIOS

2) Dado um número inteiro não negativo, crie uma função recursiva que escreva o mesmo na tela em vertical.

Entrada: 1234

Saída:

1

2

3

4

```
void printVertical(int n){  
    int cauda;  
    if (n < 10)  
        printf("\n%i",n);  
    else{  
        cauda = (int)(n/10);  
        printVertical(cauda);  
        printf("\n%i",n%10); }  
}
```


EXERCÍCIOS

3) Dado um número inteiro (negativo ou não) escreva o mesmo na vertical.

Exemplo: -1234

Saída:

-

1

2

3

4

EXERCÍCIOS

3) Dado um número inteiro (negativo ou não) escreva o mesmo na vertical.

Exemplo: -1234

Saída:

-

1

2

3

4

```
void printVertical(int n){
    int cauda;
    if (n<0){
        printf("\n-");
        printVertical(abs(n));
    }
    else if (n < 10)
        printf("\n%i",n);
    else{
        cauda = (int)(n/10);
        printVertical(cauda);
        printf("\n%i",n%10); }
}
```

EXERCÍCIOS

4) Escreva uma função recursiva, $\text{pot}(x,y)$, que devolva x elevado a potência y .

$$x^y = x \cdot x^{y-1}$$

EXERCÍCIO 4

$$x^y = x \cdot x^{y-1}$$

```
int pot(int x, int y){  
    if (y==0)  
        return 1;  
    else  
        return x*pot(x,y-1);  
}
```

EXERCÍCIO 4

```
int pot(int x, int y){  
    if (y==0)  
        return 1;  
    else  
        return x*pot(x,y-1);  
}
```

```
int b, e;  
printf("\nDigite um valor para a base: ");  
scanf("%i", &b);  
printf("\nDigite um valor para o expoente: ");  
scanf("%i", &e);  
printf("\nO valor de %d elevado a %d eh: %d\n", b, e, pot(b,e));
```

EXERCÍCIOS

5) Faça uma função recursiva para o cálculo da potência, sendo:

$$x^n = x \cdot x^{n-1}$$

Assuma que a potência será inteira, positiva ou **negativa**.

EXERCÍCIOS

5) Faça uma função recursiva para o cálculo da potência, sendo:

$$x^n = x \cdot x^{n-1}$$

Assuma que a potência será inteira, positiva ou **negativa**.

```
float pot(int x, int y){  
    if (y==0)  
        return 1;  
    if (y<0)  
        return (float)1/x * 1/pot(x, abs(y)-1);  
    return x*pot(x,y-1);  
}
```

EXERCÍCIOS

6) Escreva uma função recursiva que faça o seguinte: leia um número; se o número for negativo, a função pára; caso contrário, a função imprime o quadrado do número e faz uma chamada recursiva a si mesma.

EXERCÍCIOS

- 6) Escreva uma função recursiva que faça o seguinte: leia um número; se o número for negativo, a função pára; caso contrário, a função imprime o quadrado do número e faz uma chamada recursiva a si mesma.

```
void ex6(){
    int n;
    printf("\n\nDigite um numero: ");
    scanf("%i", &n);
    if (n >= 0){
        printf("Quadrado: %i",n*n);
        ex6();
    }
}
```

EXERCÍCIOS

7) Escreva uma função recursiva

`ImprimeSerie(i, j, k)`

que imprime na tela a série de valores do intervalo $[i,j]$, com incremento k .

EXERCÍCIOS

7) Escreva uma função recursiva, `ImprimeSerie(i, j, k)`, que imprime na tela a série de valores do intervalo `[i,j]`, com incremento `k`.

```
void ImprimeSerie(int i, int j, int k){  
    if (i <= j){  
        printf("\n%i",i);  
        ImprimeSerie(i + k, j, k);  
    }  
}
```

EXERCÍCIOS

8) Escreva uma função recursiva
 `SomaSerie(i, j, k)`
que devolva a soma da série de valores do
intervalo $[i, j]$, com incremento k .

EXERCÍCIOS

8) Escreva uma função recursiva, SomaSerie(i, j, k), que devolva a soma da série de valores do intervalo [i,j], com incremento k.

```
int SomaSerie(int i, int j, int k){  
    if (i > j)  
        return 0;  
    else  
        return i + SomaSerie(i+k, j, k);  
}
```

EXERCÍCIOS

9) Faça uma função recursiva que calcule o valor da série S descrita a seguir para um valor $n > 0$ a ser fornecido como parâmetro para a mesma.

$$S = 2 + 5/2 + 10/3 + \dots + (1 + n^2)/n$$

EXERCÍCIOS

9) Faça uma função recursiva que calcule o valor da série S descrita a seguir para um valor $n > 0$ a ser fornecido como parâmetro para a mesma.

$$S = 2 + 5/2 + 10/3 + \dots + (1 + n^2)/n$$

```
float Serie(int n){  
    if (n == 1)  
        return 2;  
    else  
        return (1.0 + n*n)/n + Serie(n - 1);  
}
```

EXERCÍCIOS

10) Faça uma função recursiva que calcule o valor da série S descrita abaixo para um valor $n > 0$.

$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$$

EXERCÍCIOS

10) Faça uma função recursiva que calcule o valor da série S descrita abaixo para um valor $n > 0$.

$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$$

```
float SerieFat(int n){  
    if (n == 1)  
        return 2;  
    else  
        return 1.0/fatorial(n) + SerieFat(n - 1);  
}
```

EXERCÍCIOS

10) Faça uma função recursiva que calcule o valor da série S descrita abaixo para um valor $n > 0$.

$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$$

```
float SerieFat(int n){  
    if (n == 1)  
        return 2;  
    else  
        return 1.0/fatorial(n) + SerieFat(n - 1);  
}
```

```
int fatorial(int n){  
    //Calcula Fatorial.  
    int fat;  
    if (n == 0)  
        fat = 1;  
    else  
        fat = n*fatorial(n-1);  
    return fat;  
}
```

$$S = 2.717$$

EXERCÍCIOS

- 11) Dado um vetor A com n elementos, construa um algoritmo recursivo para encontrar o maior elemento das entradas:
 $A[0], A[1], A[2], \dots, A[n-1]$

EXERCÍCIOS

11) Dado um vetor A com n elementos, construa um algoritmo recursivo para encontrar o maior elemento das entradas:

A[0], A[1], A[2], ..., A[n-1]

```
int num_maior(int A[], int i){
    int maior;
    if (i > 0){
        maior = num_maior(A,i-1);
        if (A[i] > maior)
            maior = A[i];
        return maior;
    }
    return A[0];
}
```

EXERCÍCIOS

11) Dado um vetor A com n elementos, construa um algoritmo recursivo para encontrar o maior elemento das entradas:

$A[0], A[1], A[2], \dots, A[n-1]$

```
int num_maior(int A[], int i){  
    int maior;  
    if (i > 0){  
        maior = num_maior(A, i-1);  
        if (A[i] > maior)  
            maior = A[i];  
        return maior;  
    }  
    return A[0];  
}
```

```
int A[7] = {10, 1, 7, 6, 4, 9, 60};  
printf("Maior = %i", num_maior(A, 7-1));
```