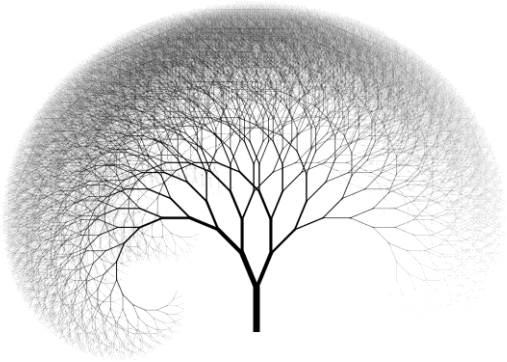
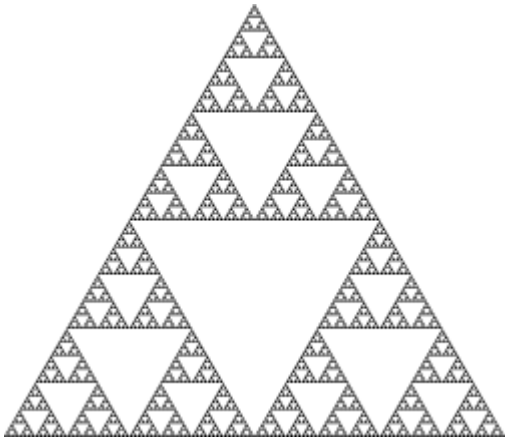
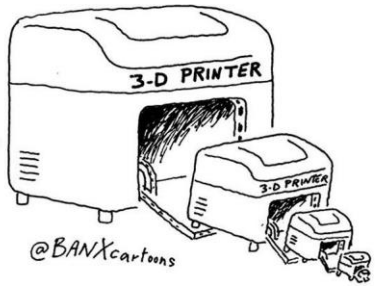
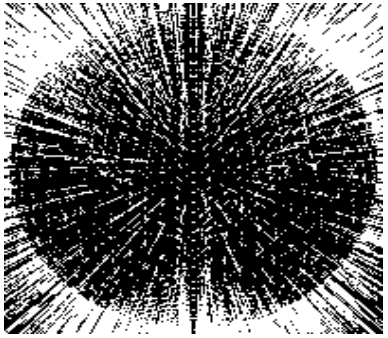
Recursividade

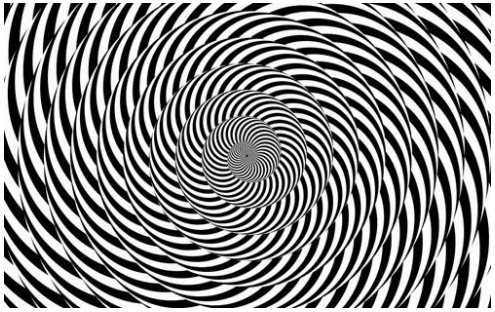


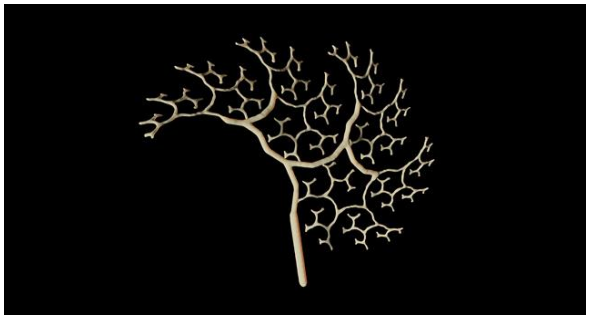




Prof. Marlus Dias Silva








UÉ, POR QUE VOCÊ ESTÁ AÍ HÁ UM TEMPO OLHANDO PRA PAREDE?
ESTOU AQUI PENSANDO EM COMO POSSO RESOLVER UM PROBLEMA DE RECURSIVIDADE
Ask Google
MUITO DIFÍCIL O PROBLEMA?
UM TANTO...
Ask Google
Ask Google
JOGUE O LIXO NO LIXO
VIDA DE PROGRAMADOR
#1516
BY NC SA

Recursividade

- A recursividade é uma estratégia que pode ser utilizada sempre que o cálculo de uma função para o valor n, pode ser descrita a partir do cálculo desta mesma função para o termo anterior (n-1).

Exemplo – Função fatorial:

$n! = n * (n-1) * (n-2) * (n-3) *....* 1$
$(n-1)! = (n-1) * (n-2) * (n-3) *....* 1$

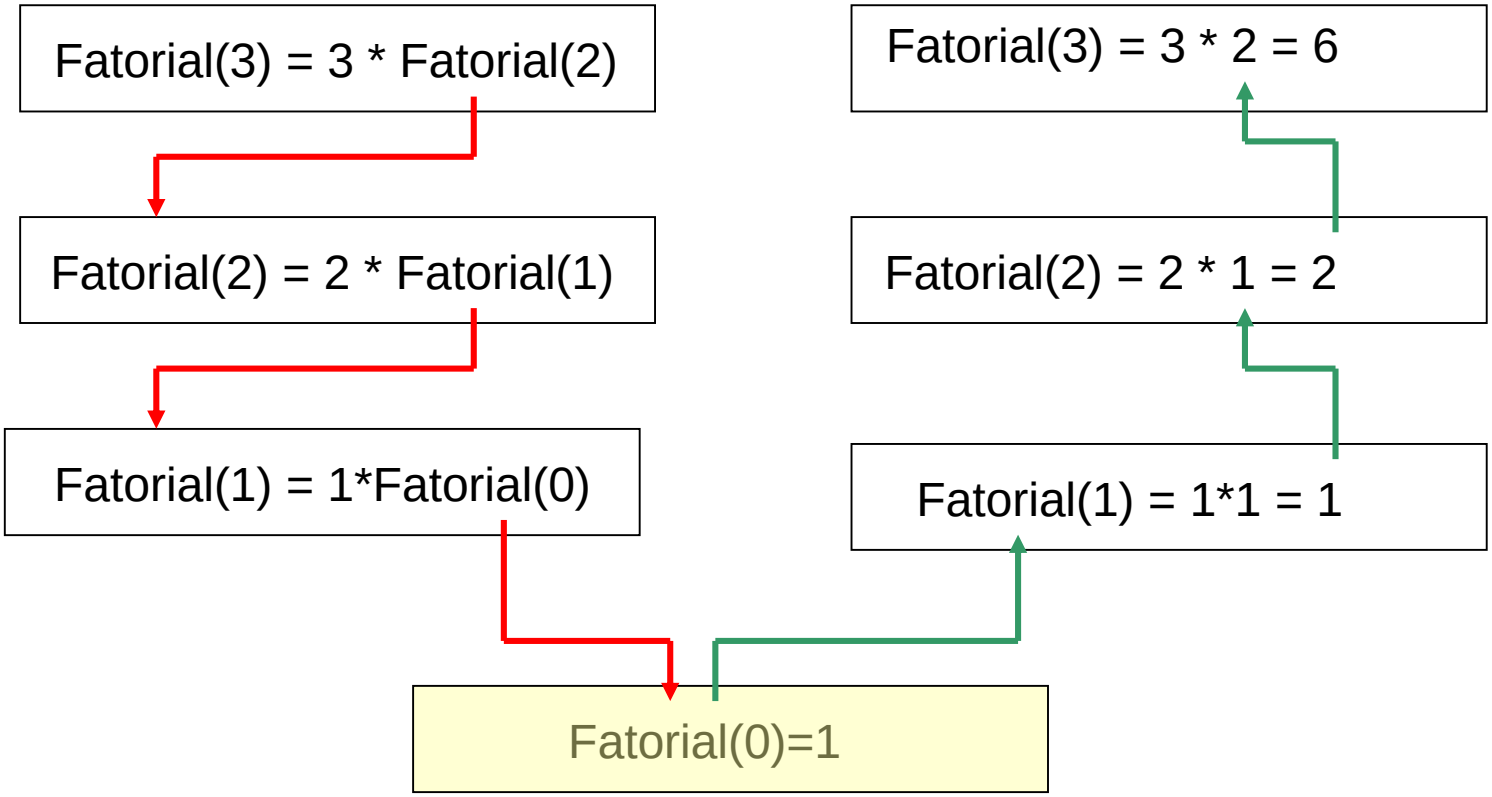
logo:

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

Definição de recursão

Note que a solução recursiva para um problema envolve um caminho de dois sentidos: primeiro o problema é decomposto no sentido top/down e depois resolvido no sentido bottom/up.

Decomposição do Fatorial(3)



Solução Iterativa

```
/* soluçao iterativa para o calculo do fatorial */

long Fatorial(int n)
{
// declarações locais
   long factN = 1;
    int i;
   for (i = 1; i <= n; i++)
       factN = factN * i;
   return factN;
} //fim fatorial
```

Solução recursiva

```
/* soluçao recursiva para o calculo do fatorial */

long Fatorial(int n)
{
// declaraçõeslocais
    if (n == 0)
          return 1;
    else
         return (n*Fatorial(n-1));
} //fim fatorial
```

Projeto para implementar funções recursivas

Toda função recursiva possui dois elementos:

- Resolver parte do problema (caso base) ou
- Reduzir o tamanho do problema (caso geral).

No caso do nosso exemplo, Fatorial(0) é o caso base

Regras para projetar uma função recursiva

- Determinar o caso base;
- Determinar o caso geral;
- Combinar o caso base e o caso geral na função.

Atenção: Cada chamada da função deve reduzir o tamanho do problema e movê-lo em direção do *caso base*. O *caso base* deve terminar sem chamar a função recursiva; isto é, executar um return.

Exemplo

Apresente um esquema recursivo para inverter uma linha de texto.

Por exemplo, inverter a linha:

TudoBem

Condição de parada: exibir o 1o. caractere - T posição 0

Parte recursiva:

exibir m e inverter texto

exibir e e inverter texto

exibir B e inverter texto

exibir o e inverter texto

exibir d e inverter texto

exibir u e inverter texto

Exibir T e parar

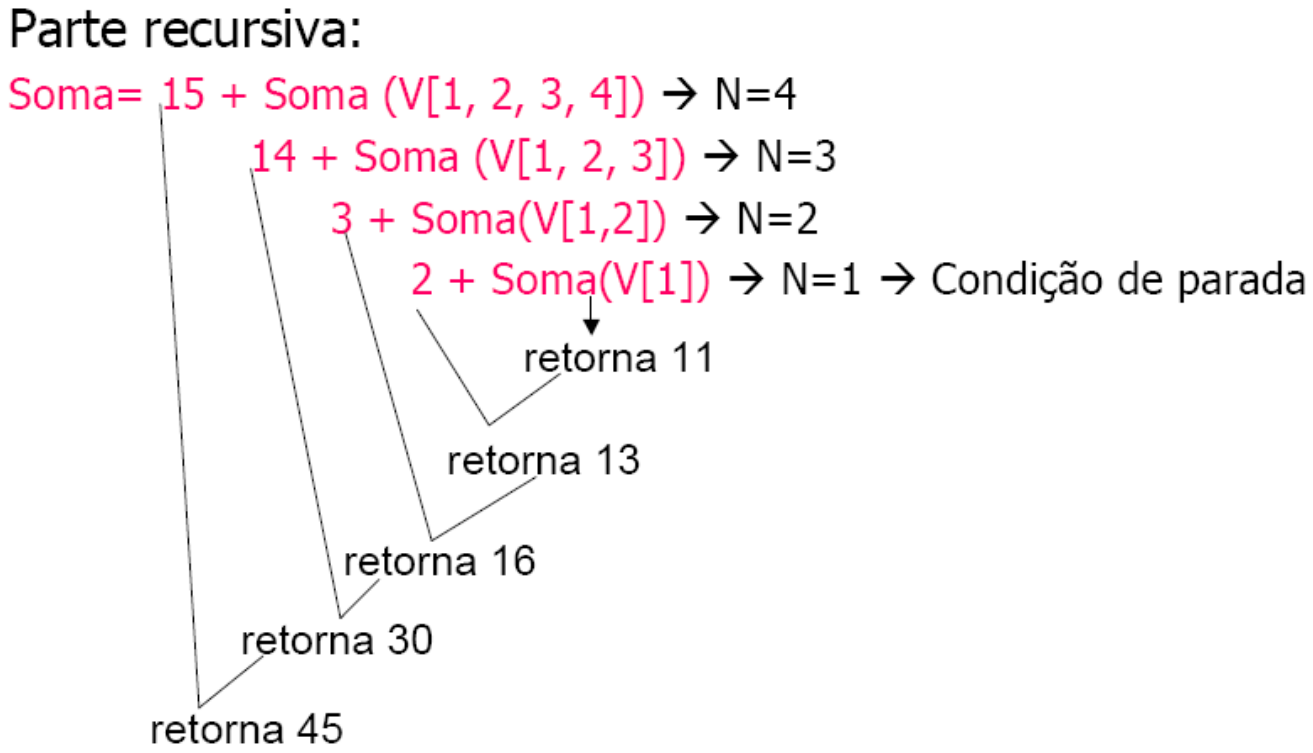
Solução

```
#include <stdio.h>
#include <string.h>

void Exibe_invertido(char mensagem[], int tamanho) {
   if (tamanho >= 0) {
        printf("%c ", mensagem[tamanho]);
        Exibe_invertido(mensagem,tamanho-1);
   }
   return;
}

int main(){

   char mensagem[10];
   int tamanho;
   printf("forneca a mensagem: ");
   scanf("%s", mensagem);
   tamanho=strlen(mensagem)-1;
   Exibe_invertido(mensagem, tamanho);
   return 0;
}
```

Exemplo

Apresente um esquema recursivo para somar os elementos de um vetor.

Por exemplo, seja V=[11, 2, 3, 14,15] e N=5



Exemplo

```
#include <stdio.h>
int Soma_elementos(int V[], int N){
   if (N ==0) return V[N];
   else return (V[N]+ Soma_elementos(V, N-1));
}

int main() {
        int i, N;
        int *V;
        printf("entre com a quantidade de numeros--> ");
        scanf("%d", &N);
        V = (int *)malloc(N*sizeof(int));
        if (!V) {
                printf("Memoria insuficiente.\n");
                exit(1);
        }
        printf("entre com os numeros--> ");
        for (i=0; i<N;i++) scanf("%d", &V[i]);
        printf("resultado=%d\n", Soma_elementos(V,N-1));
        return 0;
}
```

Números de Fibonacci

Números de Fibonacci são uma série na qual cada número é a soma dos dois números anteriores:

0, 1, 1, 2, 3, 5, ...

Para iniciar a série é preciso conhecer os dois primeiros números.

Generalização da série de Fibonacci

Dados (caso base):

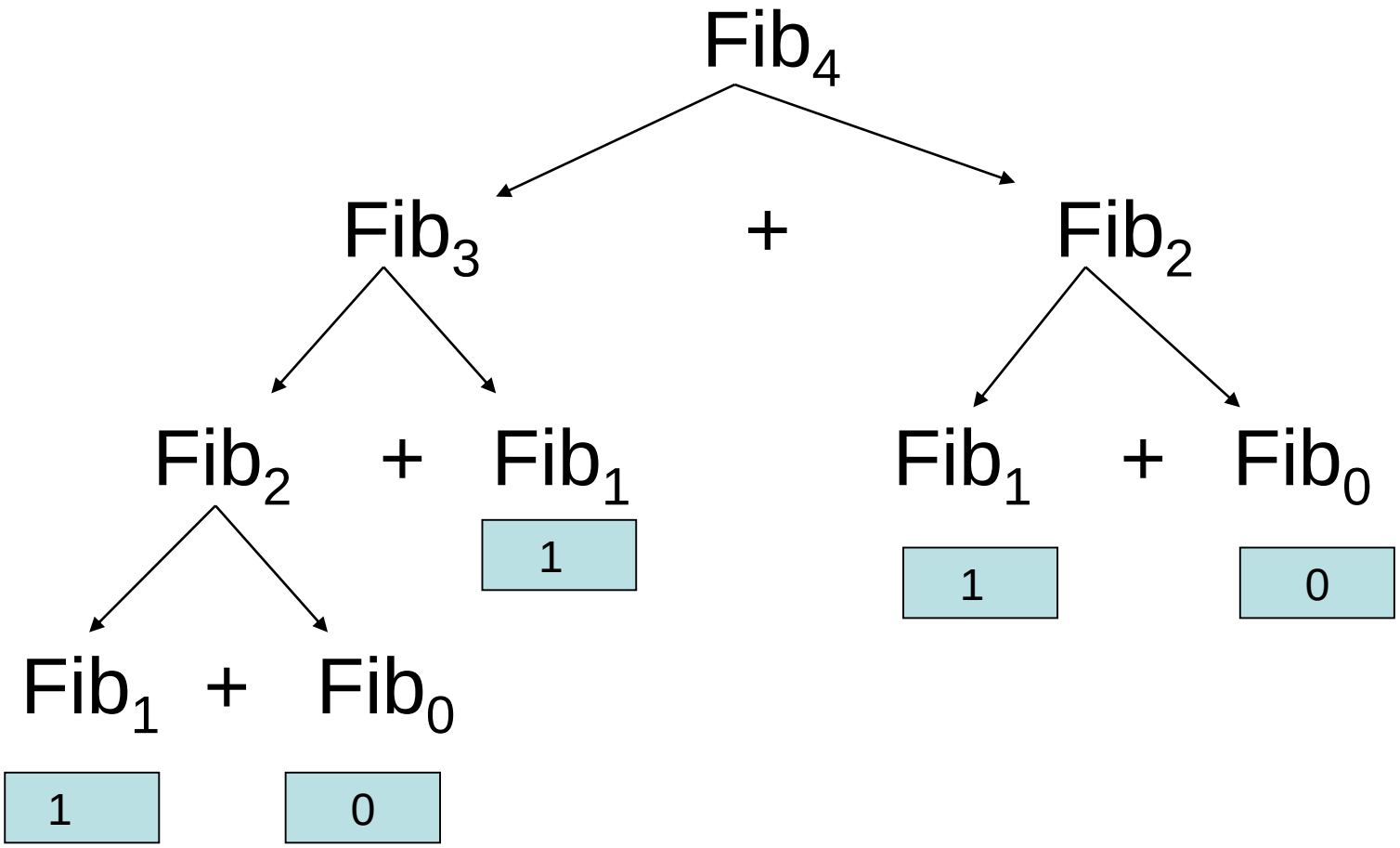
$Fibonacci_0 = 0;$

$Fibonacci_1 = 1;$

Então (caso geral):

$Fibonacci_n = Fibonacci_{n-1} + Fibonacci_{n-2}$

Generalização de $Fibonacci_4$



Exercício

1) Escreva uma função que determine a serie de Fibbonacci com n termos.

```
Long fib(int n)
{
    if (n == 0 || n == 1)  // caso base
        return n;
    return (fib(n-1) + fib(n-2)); // caso geral
}
```

Limitações da recursão

- Soluções recursivas podem envolver alto overhead devido à chamada de funções;
- A cada chamada da função recursiva, espaço de memória (na pilha) é alocada. Se o número de chamada recursiva é muito grande, pode não ter espaço suficiente na pilha para executar o programa (segment fault)