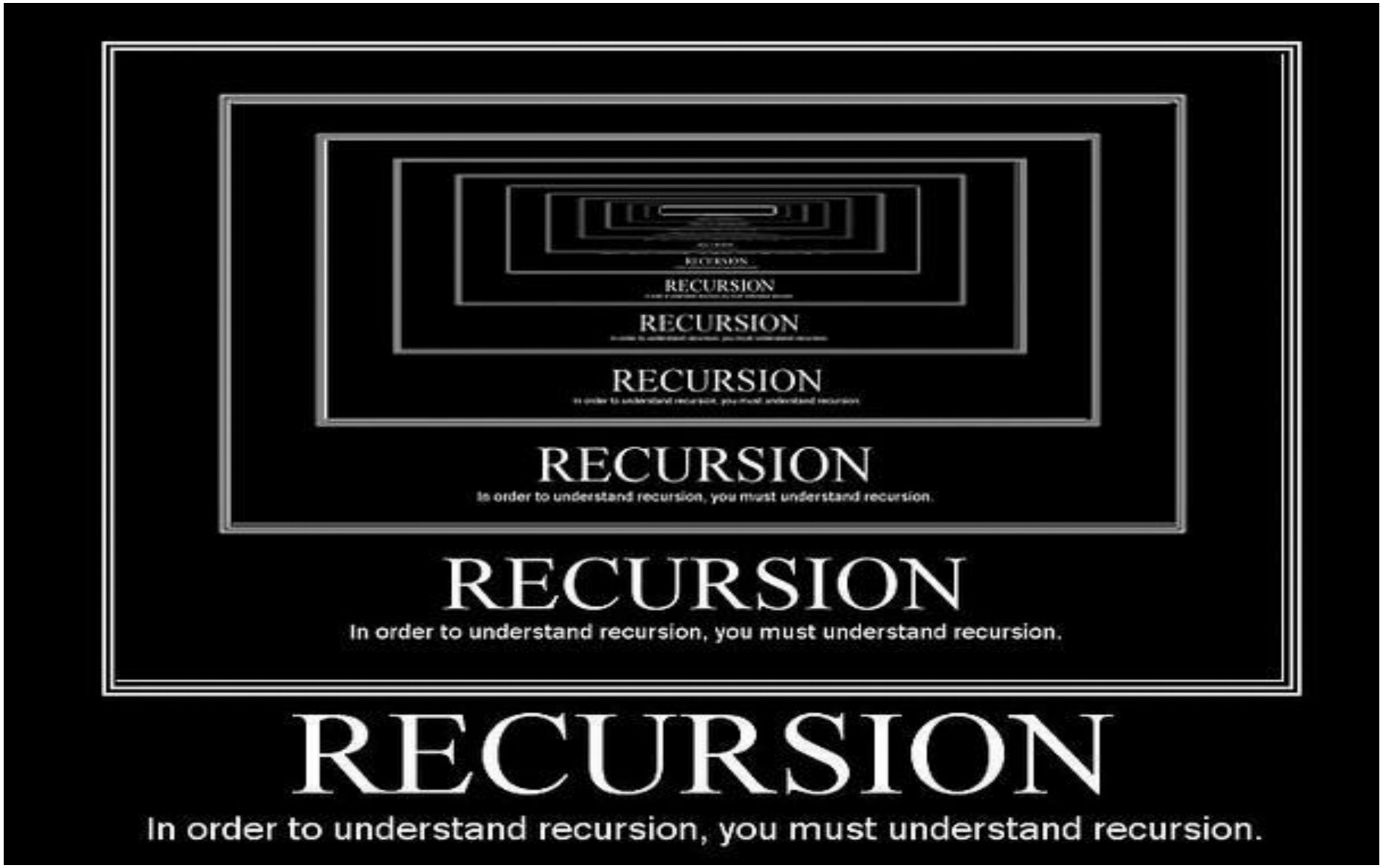
SIN110 – Programação

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Recursividade

Recursividade



Para entender a recursão, você deve entender a recursão.

Recursividade

Definições:

Os programas que discutimos geralmente são estruturados como funções que chamam umas às outras de uma maneira hierárquica, disciplinada.

O que é útil para alguns problemas.

Recursividade

Definições:

Recursividade é uma técnica particularmente poderosa em definições matemáticas, cujo poder deve-se à possibilidade de se **definir um conjunto infinito** de objetos **através de uma formulação finita**.

Recursividade

Definições:

Em termos de programação, a recursividade aparece em funções chamadas **funções recursivas**.

Uma **<u>função recursiva</u>** é uma função que chama a si mesma.

O uso da recursividade geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema a ser resolvido é recursivo por natureza.

Funções em C

Definição:

```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

- tipo_da_função: refere-se ao tipo de resposta que a função devolve
- nome_da_funcao: é o identificador da função
- lista_de_parametros: é uma lista de variáveis que representam valores de entrada para a função

Recursividade

No Brasil também temos um produto com figura recursiva.

Recursividade

Exemplo 1:

Como se calcula o fatorial de 4?

$$4! = 4 * 3 * 2 * 1 = 24$$

$0! = 1$	$0! = 1$
$1! = 1$	$1! = 1 * 0!$
$2! = 2 * 1$	$2! = 2 * 1!$
$3! = 3 * 2 * 1$	$3! = 3 * 2!$
$4! = 4 * 3 * 2 * 1$	$4! = 4 * 3!$

Recursividade

Exemplo 1:

Definição da função fatorial:

$$N! = N * (N-1) * (N-2) * \ldots * 1$$

Escrevendo de uma outra forma:

$$F(n) = \begin{cases} 1, & n = 0 \\ n * F(n-1) & n > 0 \end{cases}$$

Recursividade

Exemplo 1:

Essa segunda definição (**Definição recursiva**) é mais precisa!

$$\mathbf{F(n)} = \begin{cases} \mathbf{1,} & \mathbf{n = 0} \\ \mathbf{n * F(n-1)} & \mathbf{n > 0} \end{cases}$$

Caso Base, que garante o fim da recursão

Passo de Recursão, função definida em função dela mesma

Recursividade

A função sabe resolver somente o(s) caso(s) mais simples, **caso(s) básico(s)**.

Se ela é chamada com um caso básico, ela simplesmente **retorna** um resultado.

Quando uma função chama a si mesma, ocorre uma **chamada recursiva** que também dito **passo de recursão** porque os resultados são combinados para formar o **resultado final**.

Recursividade

No caso da computação, os **procedimentos recursivos** fazem chamadas a eles mesmos.

E como no caso da definição de funções matemáticas recursivas, um **algoritmo recursivo** deve possuir algo que o impeça de se chamar **indefinidamente,** caso contrário, nunca parará até que se esgote os recursos da máquina.

A parte que garante o fim da recursividade é justamente o **caso base** ou **caso trivial**.

Recursividade

Voltando ao exemplo 1 (cálculo do fatorial):

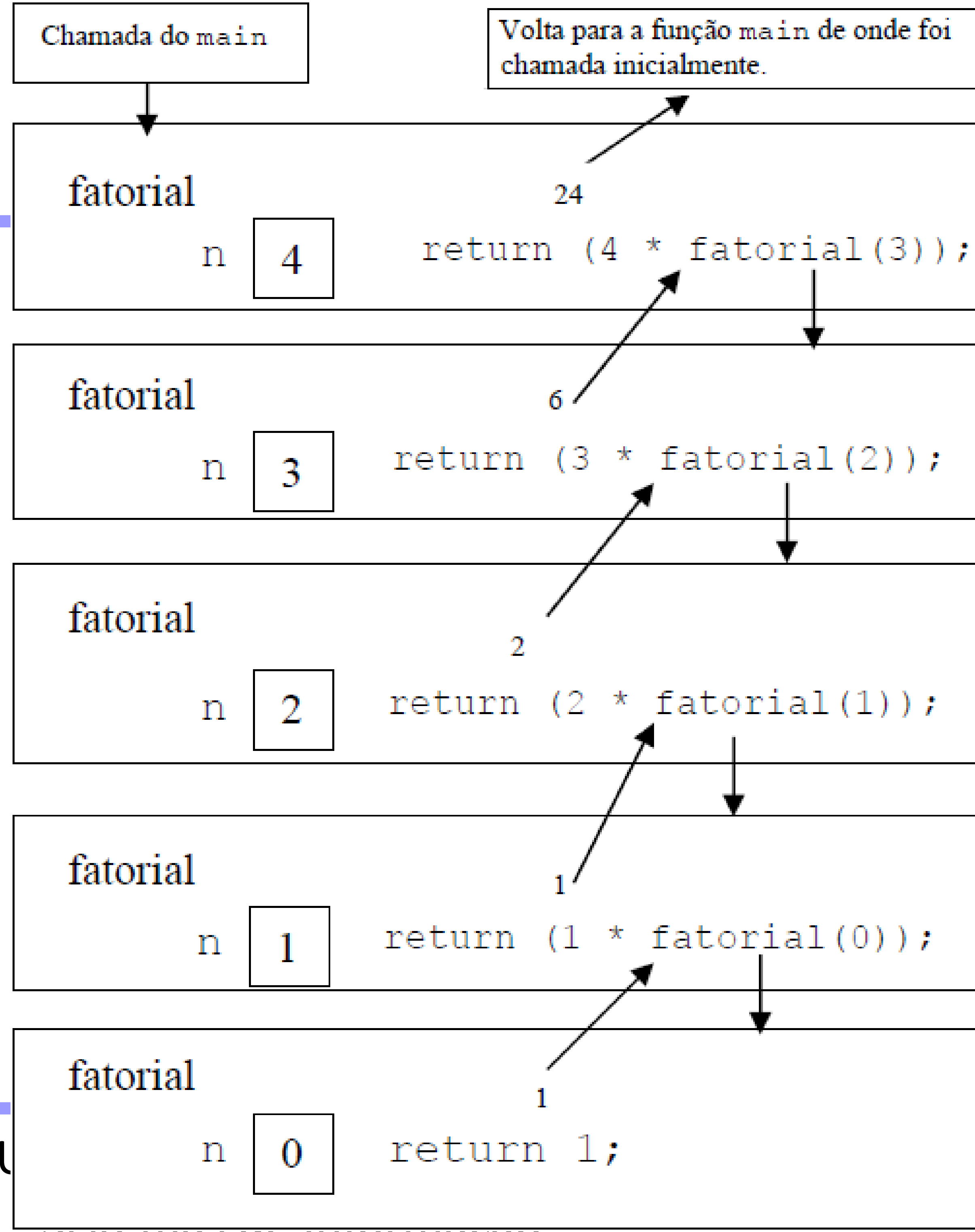
$$F(n) = \begin{cases} 1, & n = 0 \\ n * F(n-1) & n > 0 \end{cases}$$

```
int fatorial (int n) {
    if (n == 0) // caso base
        return 1;
    else
        return (n * fatorial (n-1));
        //passo recursivo
}
```

Recursividade

Voltando ao exemplo 1 (cálculo do fatorial):

Vejamos um esquema ilustrativo do que ocorre na memória quando a versão recursiva é chamada passando-se 4. Suponha que fatorial seja chamada da função main.



Recursivo X Iterativo

Exemplo 1:

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

Recursivo

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

Iterativo

Recursivo X Iterativo

Os programas recursivos, em geral, são mais lentos do que seus equivalentes *iterativos*.

Isto ocorre porque a chamada de uma função é uma operação lenta (necessidade de empilhar os argumentos, endereço de retorno, variáveis locais, etc).

No entanto, existem alguns problemas onde a solução recursiva é muito mais prática.

Recursividade

Uma função recursiva deve **obrigatoriamente** ter um critério de parada.

A **parada da recursividade** se dá pelo caso base (que não possui recursão).

Recursividade

Outro exemplo clássico de recursividade:

A série de Fibonacci:

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

Qual a propriedade desta sequência?

Recursividade

Outro exemplo clássico de recursividade:

A série de Fibonacci inicia com **0 e 1** e tem a propriedade de que cada número de Fibonacci subsequente é **a soma dos dois anteriores**.

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

Recursividade

Outro exemplo clássico de recursividade:

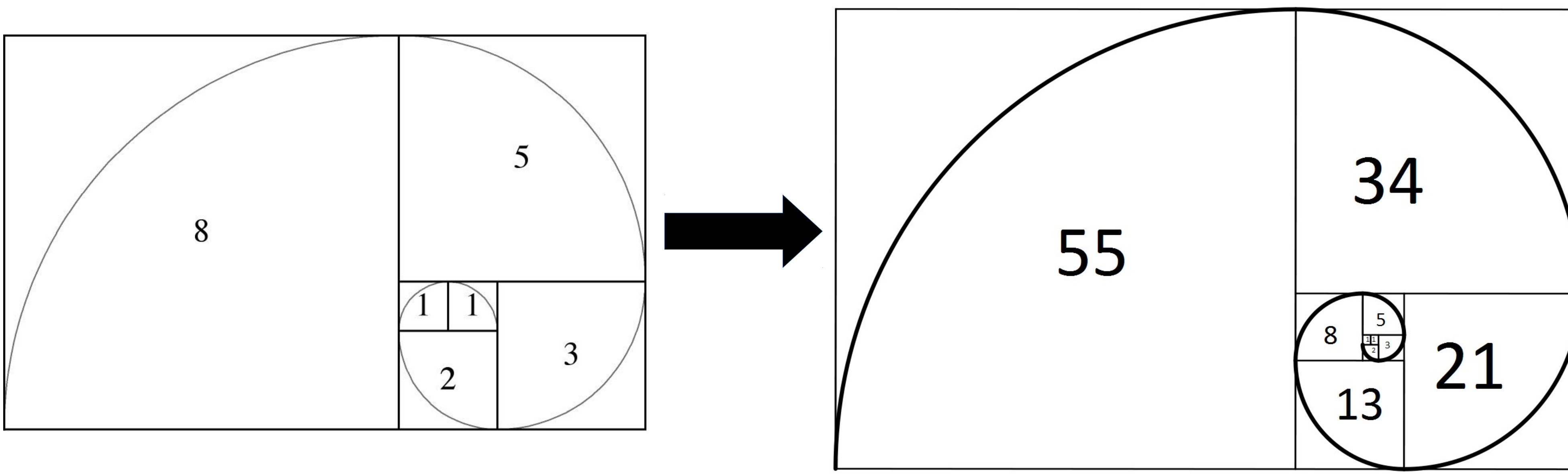
A série de Fibonacci inicia com **0 e 1** e tem a propriedade de que cada número de Fibonacci subsequente é **a soma dos dois anteriores**.

```
int fibonacci (int n) {
    if (n <= 1)  //caso base
        return n;
    return (fibonacci(n-2) + fibonacci(n-1));
    //passo da recursão
}
```

Recursividade

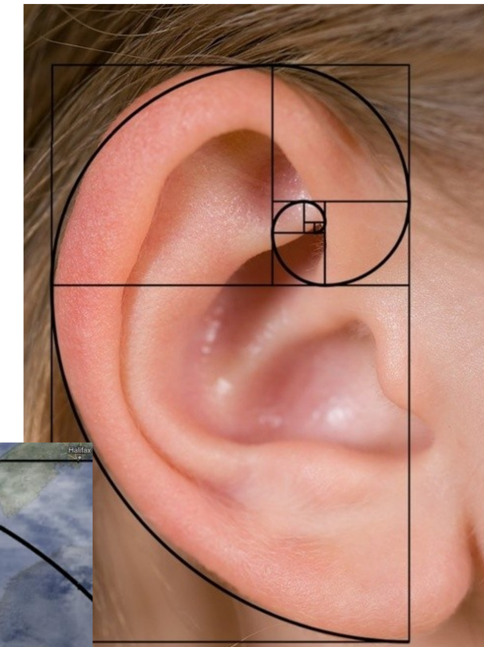
Fibonacci curiosidades:

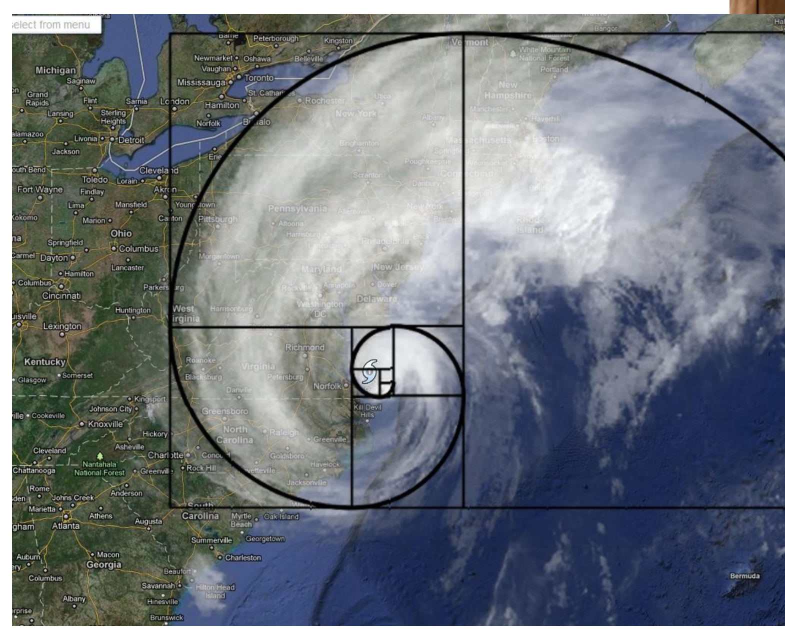
Os números da série definem uma espiral.



Recursividade

Fibonacci curiosidades: Na natureza





SIN110 – Programação

Recursividade

Fibonacci Iterativo:

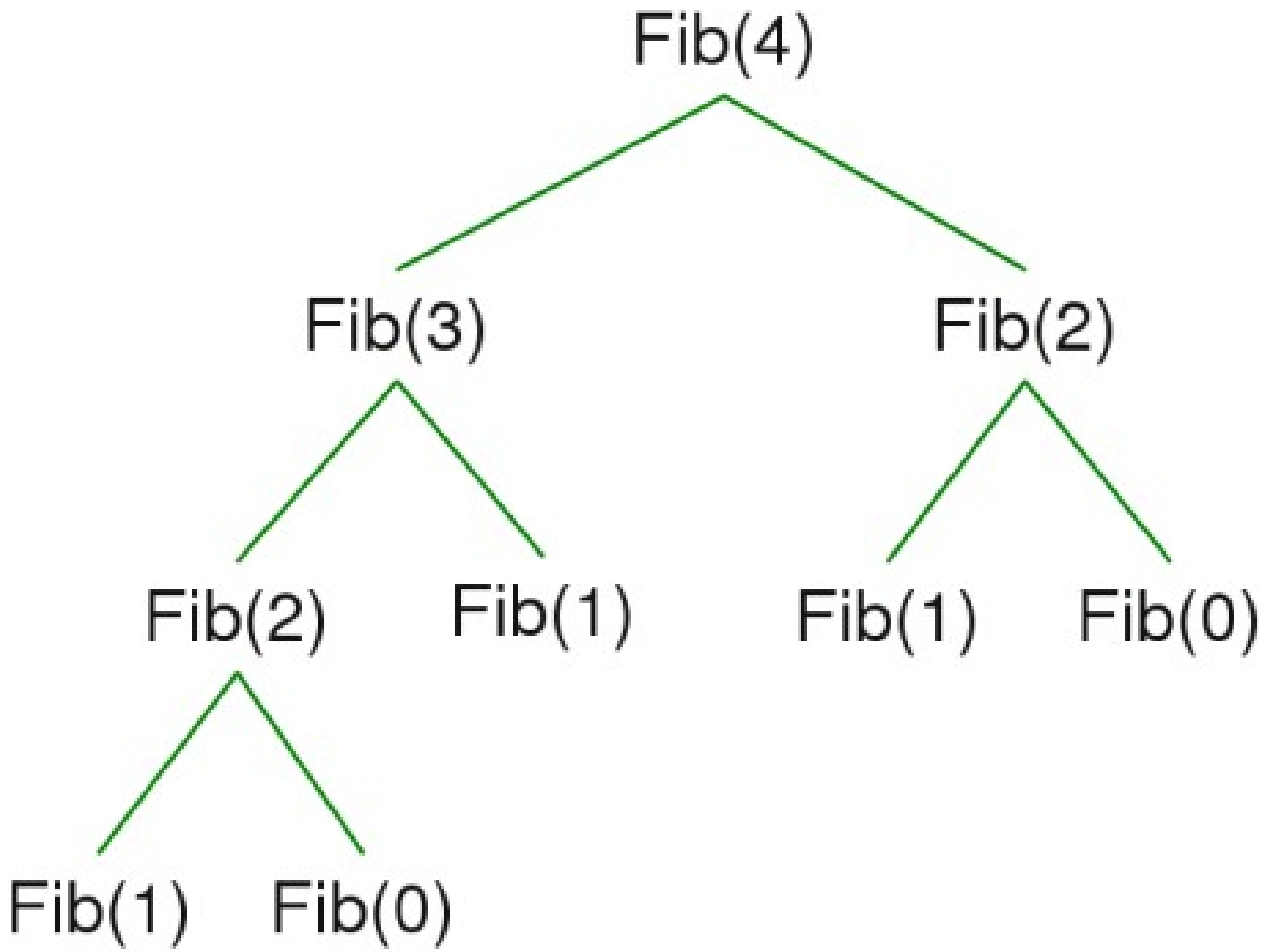
```
int fibonacci (int n) {
   if(n<=1)
       return n;
    int f1=0, f2=1, f3, i;
    for(i=2;i<=n; i++){
       f3 = f1+f2;
       f1 = f2;
       f2 = f3;
    }
    return f3;
}
```

Recursividade

```
int fibonacci (int n) {
    if (n <= 1)  //caso base
        return n;
    return (fibonacci(n-2) + fibonacci(n-1));
    //passo da recursão
}
```

A solução recursiva é direta e mais simples que a solução iterativa. Contudo, apresenta uma **armadilha grave** que deve ser levada em conta. Observe que a função refere-se a si mesma 2 vezes: fibonacci(n-1) e fibonaccir(n-2) e, neste caso, faz muitos cálculos repetidos.

Recursividade



Recursividade

No cálculo de fibonacci(100), por exemplo, a função necessita calcular independentemente fibonacci(99) e fibonacci(98).

Para calcular fibonacci(99), a função vai calcular fibonacci(98) e fibonacci(97) independente de ele já ter sido calculado antes.

Assim, uma série de cálculos repetidos serão feitos.

Recursividade

Exemplo 3: (Cálculo do Somatório num intervalo)

Qual o somatório de [2, 5]?

Somatorio(2, 5) = 2 + 3 + 4 + 5 = 14

Recursividade

Exemplo 3: (Cálculo do Somatório num intervalo)

```
int somatorio (int m, int n) {
   if (m == n)  // caso base
      return m;
   else
      return (m + somatorio (m+1, n));
      //passo recursivo
}
```

Dado S(m,n), onde n > m, calcule S(2 , 5):

S(2, 5) =2 + S(3, 5)

S(3, 5) = 3 + S(4, 5)

S(4,5) = 4 + S(5,5)

S(5,5) = 5

Recursividade

Exemplo 3: (Cálculo do Somatório num intervalo)

Representação matemática:

$$\sum_{k=m}^{n} = \begin{cases} m & \text{se } n = m \text{ e} \\ m + \sum_{k=m+1}^{n} & \text{se } n > m. \end{cases}$$

```
int somatorio (int m, int n) {
    if (m == n)  // caso base
        return m;
    else
        return (m + somatorio (m+1, n));
        //passo recursivo
}
```

Recursividade

Exemplo 3: (Cálculo do Somatório num intervalo)

```
int somatorio (int m, int n) {
   if (m == n)  // caso base
      return m;
   else
      return (m + somatorio (m+1, n));
      //passo recursivo
}
```

Como seria o código iterativo dessa função?

Recursividade

Exemplo 4: (Cálculo da Potência)

Representação matemática:

$$x^n = \begin{cases} \left(\dfrac{1}{x}\right) \times x^{n+1} & se\ n < 0 \\ 1 & se\ n = 0 \\ x \times x^{n-1} & se\ n > 0 \end{cases}$$

Recursividade

Exemplo 4: (Cálculo da Potência)

Quanto é potencia(3, 4)? 3^4

Dado que:

$$
\left.
\begin{aligned}
3^0 &= 1 \\
3^1 &= 3 \\
3^2 &= 3 * 3 \\
3^3 &= 3 * 3 * 3 \\
3^4 &= 3 * 3 * 3 * 3
\end{aligned}
\right\}
\qquad
\begin{aligned}
3^0 &= 1 \\
3^1 &= 3 * 3^0 \\
3^2 &= 3 * 3^1 \\
3^3 &= 3 * 3^2 \\
3^4 &= 3 * 3^3
\end{aligned}
$$

Recursividade

Exemplo 4: (Cálculo da Potência)

```
int potencia(int x, int y){
        if(y==0) // caso base
           return 1;
        else if(y>0)
              return x*potencia(x, y-1);
              //passo recursivo
}
```

Recursividade

Exemplo 4: (Cálculo da Potência)

```
float potencia(float x, float y){
   if(y==0) // caso base
      return 1;
   else if(y>0)
          return x*potencia(x, y-1);
          //passo recursivo
   else if(y<0)
          return (1/x)*potencia(x, y+1);
          //passo recursivo
}
```

Exercícios

1) Escreva uma função recursiva que mostre na tela os números inteiros de 1 a 5 em ordem crescente. Outra para imprimir em ordem decrescente.

2) Escreva uma função recursiva que imprima os elementos de um vetor de letras em ordem decrescente.

Ex. 1

```c
#include <stdio.h>

void imprimeCrescente(int ini, int fim){
	if(ini <= fim){
		printf("%d ", ini);
		imprimeCrescente(ini+1, fim);//Passo Recursivo
	}
	return;//Fim da Recursão
}

void imprimeDecrescente(int ini, int fim){
	if(ini <= fim){
		imprimeDecrescente(ini+1, fim);//Passo Recursivo
		printf("%d ", ini);
	}
	return;//Fim da Recursão
}

int main(){

	imprimeCrescente(1, 5);
	printf("\n");

	imprimeDecrescente(1, 5);
	printf("\n");

	system("pause");
	return 0;
}
```

Ex. 2

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

void imprimeDecrescente(char *vet, int ini, int fim){
    if(ini <= fim){
        imprimeDecrescente(vet, ini+1, fim);//Passo Recursivo
        printf("%c", vet[ini]);
    }
    return;//Fim da Recursão
}

int main(){

    char vetor[100];

    gets(vetor);
    imprimeDecrescente(vetor, 0, strlen(vetor)-1);
    printf("\n");


    system("pause");
    return 0;
}
```

Exercícios

3) Escreva uma função recursiva que retorne a soma dos números de um intervalo partindo-se sempre do maior número e terminando no menor.

4) Escreva uma função recursiva que faça a multiplicação de dois números inteiros positivos de forma recursiva. **N*M = M + M + M + … + M (N Vezes)**.

Ex. 3

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

int somaIntervalo(int ini, int fim){
    if(ini < fim){
       printf("%d + ", fim);
       return fim + somaIntervalo(ini, fim-1);//Passo Recursivo
    }
    if(ini > fim){
       printf("%d + ", ini);
       return ini + somaIntervalo(ini-1, fim);//Passo Recursivo
    }
    if(ini == fim){//Fim da Recursao
       printf("%d = ", ini);
       return ini;
    }
}

int main(){

    int ini, fim, soma;
    printf("Digite dois numeros:\n");
    scanf("%d %d", &ini, &fim);

    printf("Somando...\n");
    soma = somaIntervalo(ini, fim);
    printf("%d\n", soma);


    system("pause");
    return 0;
}
```

Ex. 4

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

int multiplicao(int n, int m){
    if(n == 0){
       return 0;//Fim da Recursao
    }else{
       return m + multiplicao(n-1, m);//Passo recursivo
    }
}

int main(){

    int N, M, resultado;
    printf("Digite dois numeros para multiplicar:\n");
    scanf("%d %d", &N, &M);

    resultado = multiplicao(N, M);

    printf("%d * %d = %d\n", N, M, resultado);

    system("pause");
    return 0;
}
```