



RECURSIVIDADE

Técnica de programação, na qual um subprograma (**função**) chama a si mesmo.

Importante: todas as **funções recursivas** devem encerrar (concluir) a recursão a partir de um teste de valor ou condição!!!



Exemplo: Fatorial de um número

$$n! = n * (n-1) * (n-2) * \dots * 1$$

Função direta

```
int fatorial (int n) // solução direta para calcular o fatorial
{
    int fat; // valor de retorno da funcao
    int aux; // variável auxiliar, para acumular o valor do fatorial
    fat = 1;
    for (aux = n; aux > 1; aux--) // se n=0 ou 1, nem entra no for
    {
        fat = fat * aux;
    }
    return fat; // encerra função, retornando valor
}
```



Fatorial de um número

$$n! = n * \underbrace{(n-1) * (n-2) * \dots * 1}_{(n-1)!}$$

Definição recursiva:

$$0! = 1;$$

$$n! = n * (n-1)!, \text{ para } n > 0.$$

Fatorial de um número

$$5! = 5 * \underbrace{4 * 3 * 2 * 1}_{4!}$$

$$= 5 * 4!$$

$$4! = 4 * \underbrace{3 * 2 * 1}_{3!}$$

$$= 4 * 3!$$

$$3! = 3 * \underbrace{2 * 1}_{2!}$$

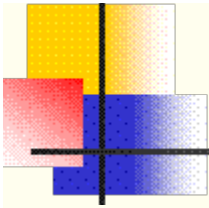
$$= 3 * 2!$$

se $n = 0$ (ou 1)

fatorial de $n \leftarrow 1$

senão

fatorial de $n \leftarrow n * \text{fatorial de } (n-1)$



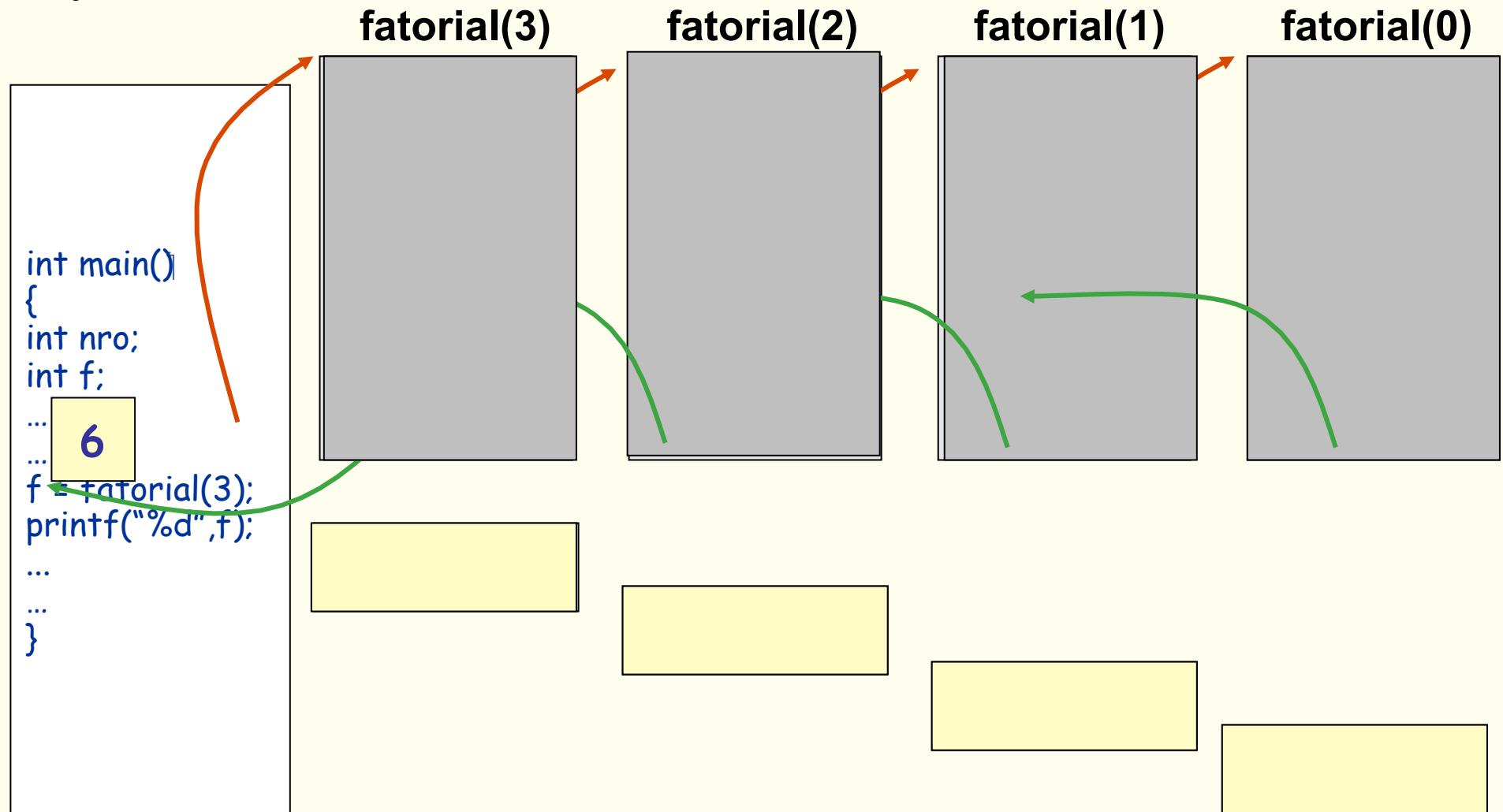
Regra de recursão

$$\begin{cases} \text{se } n = 0 \text{ então fatorial de } n \leftarrow 1 \\ \text{se } n > 0 \text{ então fatorial de } n \leftarrow n * (\text{fatorial de } n-1) \end{cases}$$

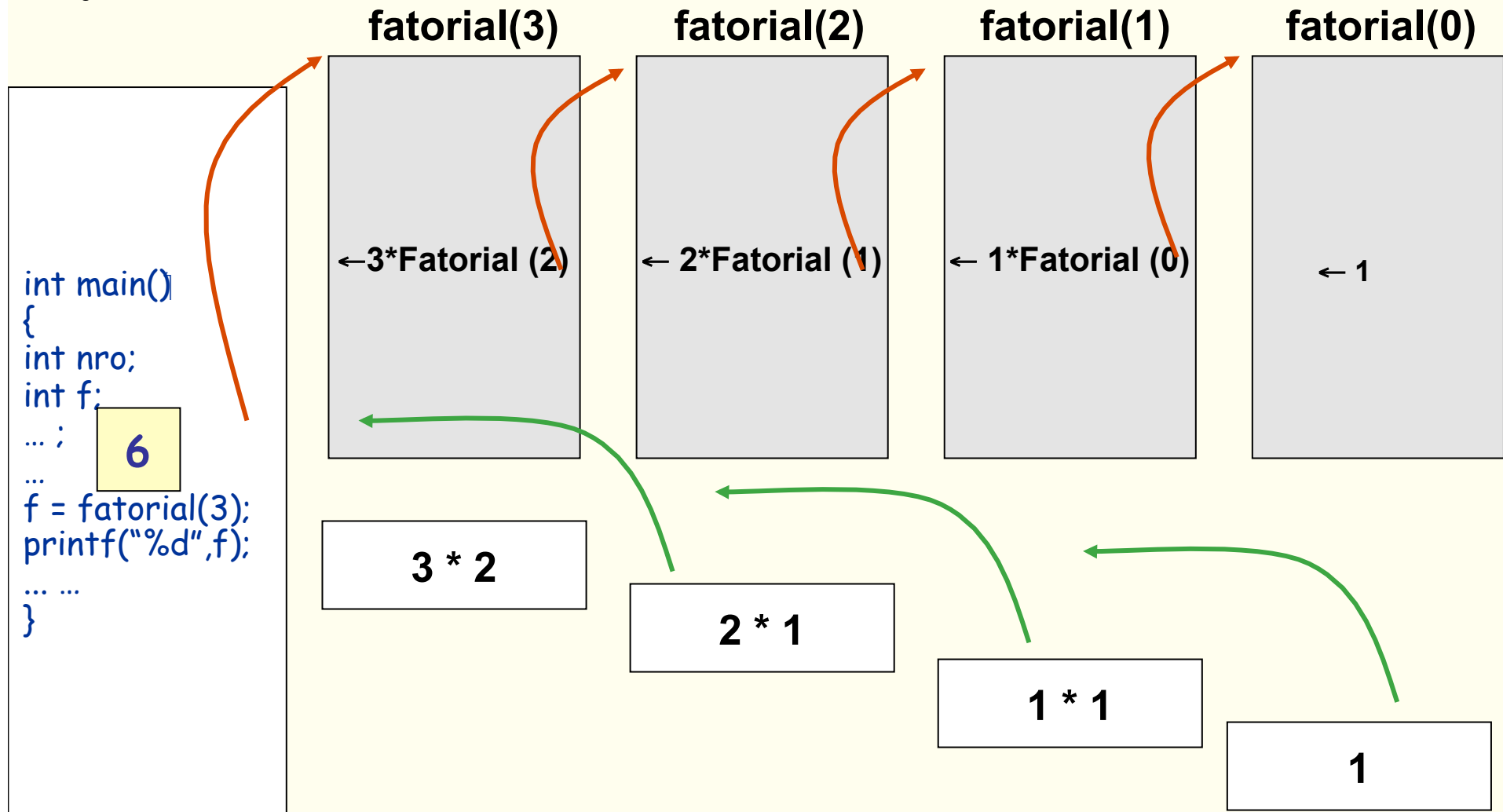
Função recursiva:

```
int fatorial (int n) // solução recursiva para o cálculo do fatorial
{
    int fat;
    if (n == 0) // encerra a recursão e inicia o retorno
        fat = 1;
    else
        fat = n * fatorial ( n - 1 ); // chamada recursiva
    return fat; // encerra função
}
```

Fatorial de um número: execução



Fatorial de um número: execução





Execução de uma chamada de um subprograma recursivo

Início da execução

- alocação de variáveis locais
- parâmetros

Processamento

- se a execução for interrompida por nova chamada recursiva, ocorre o “empilhamento” dos indicadores de execução atuais (valores, ponto de retorno)

Fim da execução

- liberação de variáveis locais - “desempilhamento”
- retorno ao ponto de chamada


```

int main( )
{
    int k;

    ...

    k=r(3);
    printf("%d",k);
    ...
}

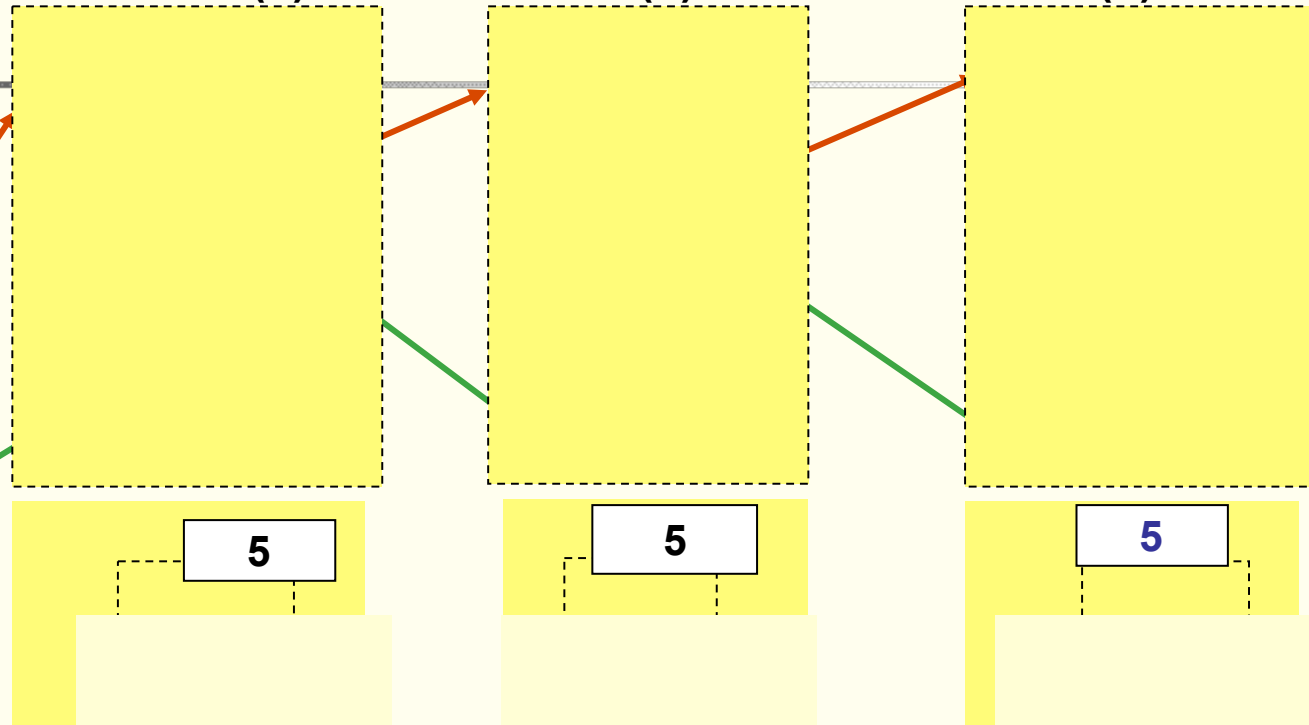
```

k 5

R (3)

r (7)

r (2)



```

int r (int p)
{
    int ret; // retorno
    int lido; //valor lido
    printf("informe valor:");
    scanf("%d",&lido);
    if (l != 5)
        ret = r(lido);
    else
        ret = lido;
    return ret;
}

```

```

int main( )
{
    int k;

    ... ;

    ...

    k = r(3);
    printf("%d",k);

    ...
}

```

k 5

r (3)

```

...
scanf("%d",&l);
//Lido:7
...
return(r(l));
...
end { r };

```

r 3
5

r (7)

```

...
scanf("%d",&l);
//Lido:2
...
return(r(l));
...
}

```

r 7
5

r (2)

```

...
scanf("%d",&l);
// Lido:5
...
return(l);
}

```

r 2
5

```

int r (int p)
{
    int ret; // retorno
    int lido; //valor lido
    printf("informe valor:");
    scanf("%d",&lido);
    if (l != 5)
        ret = r(lido);
    else
        ret = lido;
    return ret;
}

```

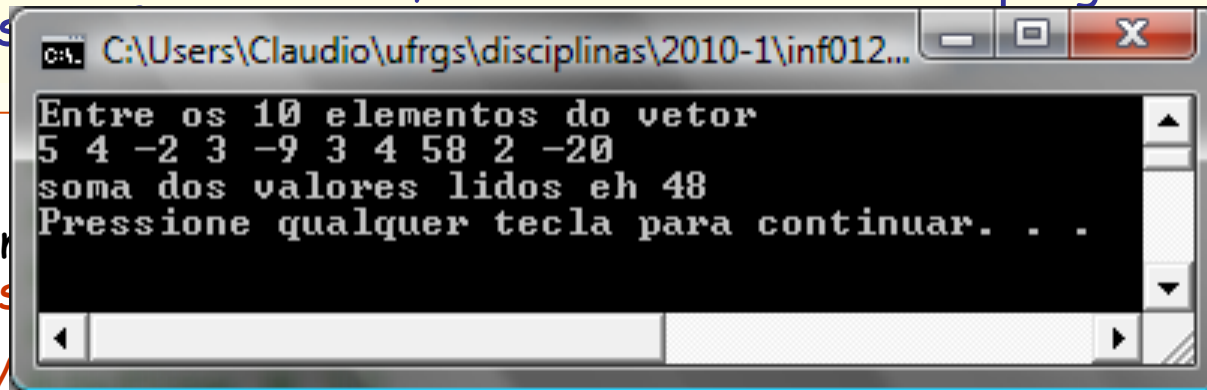
Exercício: fazer um programa contendo uma função que calcula a soma dos n elementos inteiros de um vetor, de forma recursiva. No programa principal, ler os números e informar a soma.

```
...
#define N 10
int soma_rec (int v[ ], int ultimo) // calcula soma dos elementos de vetor de
    forma recursiva
{
    int soma; // retorno da funcao
    if (ultimo == 0) // se o vetor tiver apenas um elemento
        soma = v[0];
    else
        soma = v[ultimo] + soma_rec (v, ultimo - 1);
    return soma;
}
int main (void)
{
    int numeros[N];
    int i;
    printf("Entre os %d elementos do vetor\n", N);
    for (i = 0; i < N; i++)
        scanf ("%d", &numeros[i]); // lê valores: incluir printf explicativo

    printf("soma dos valores lidos eh %d\n", soma_rec(numeros, N-1) );
    system("pause");
    return 0;
}
```

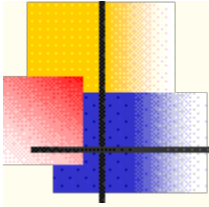
Exercício: fazer um programa contendo uma função que calcula a soma dos n elementos inteiros de um vetor, de forma recursiva. No programa principal, ler os números

```
...  
#define N 10  
int soma_rec (int v[], int ultimo) {  
    // forma recursiva  
    int soma; // soma dos elementos do vetor  
    if (ultimo == 0) // se o vetor tiver apenas um elemento  
        soma = v[0];  
    else  
        soma = v[ultimo] + soma_rec (v, ultimo - 1);  
    return soma;  
}  
int main (void)  
{  
    int numeros[N];  
    int i;  
    printf("Entre os %d elementos do vetor\n", N);  
    for (i = 0; i < N; i++)  
        scanf ("%d", &numeros[i]); // lê valores: incluir printf explicativo  
  
    printf("soma dos valores lidos eh %d\n", soma_rec(numeros, N-1) );  
    system("pause");  
    return 0;  
}
```



```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf012...  
Entre os 10 elementos do vetor  
5 4 -2 3 -9 3 4 58 2 -20  
soma dos valores lidos eh 48  
Pressione qualquer tecla para continuar. . .
```

for de



Sequência de Fibonacci

Descoberta por Leonardo Pisano - 1202

1 1 2 3 5 8 13 ...

$$\text{fib}(1) = 1$$

$$\text{fib}(2) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2), \text{ para } n > 2$$



Cálculo do $n^{\text{ésimo}}$ termo

```
int fib (int n) // devolve apenas o n-ésimo termo, solução direta
{
    // devolve o termo n da sequência de Fibonacci
    int i;
    int fibn= 1, fibn_1 = 0, fibn_2 = 1; //termos fib(n), fib(n-1) e fib(n-2))
    for (i=1; i <= n; i++)
    { //calcula TODOS os termos, inclusive os 2 primeiros:
        fibn = fibn_2 + fibn_1;
        fibn_2 = fibn_1; //atualiza os antecessores na série
        fibn_1 = fibn;
    }
    return fibn; /* retorna o termo solicitado*/
}
```



Sequência de Fibonacci

1 1 2 3 5 8 13 ...

FIB (5)?

$$\text{fib}(1) = 1$$

$$\text{fib}(2) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2), \text{ para } n > 2$$

recursão!

Sequência de Fibonacci

1 1 2 3 5 8 13 ...



FIB (5)

FIB (4)

+

FIB (3)

FIB (3) + FIB (2)

FIB (2) + FIB (1)

FIB (2) + FIB (1)

1

1

1

1

1



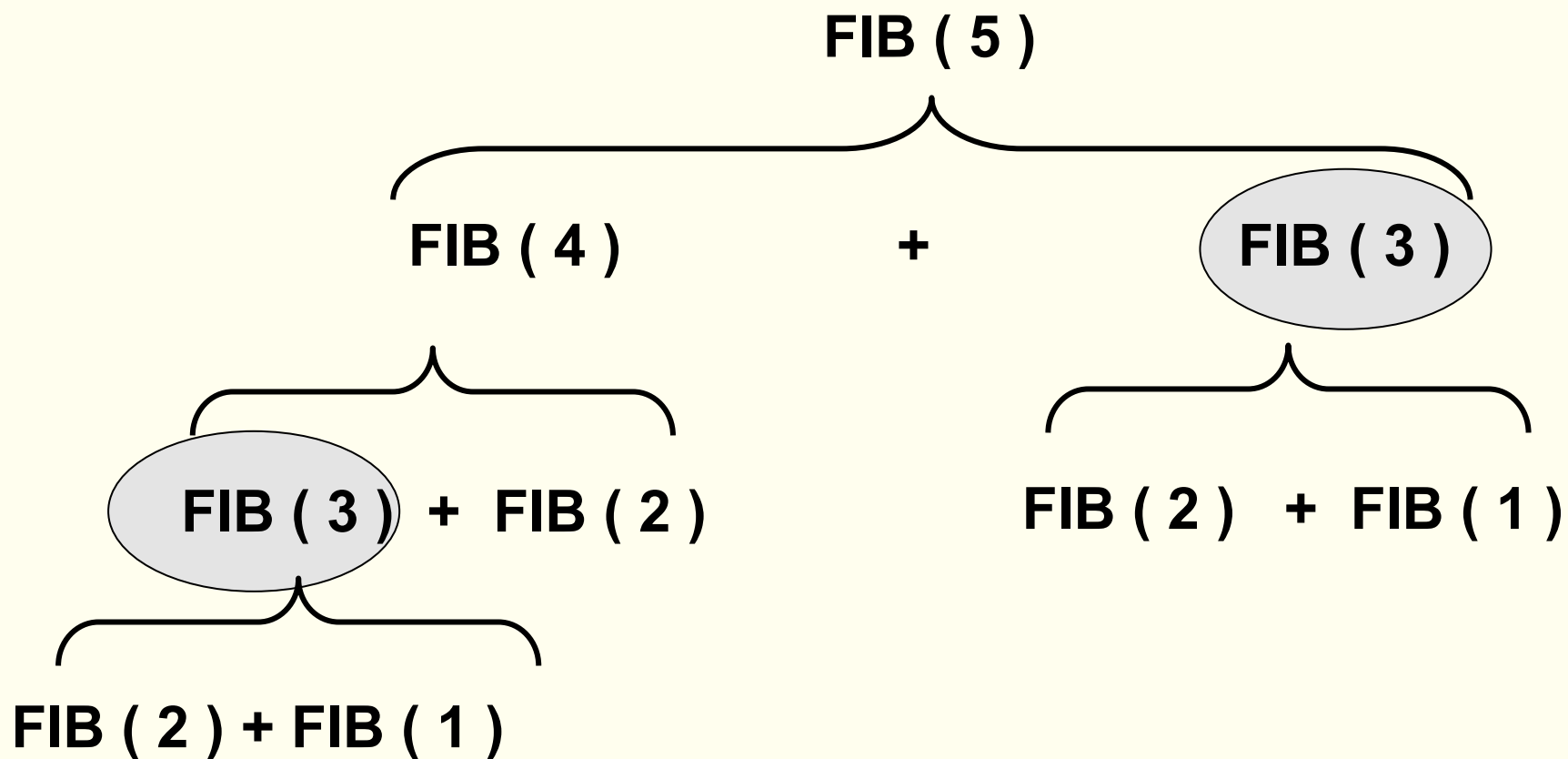
Sequência de Fibonacci

1 1 2 3 5 8 13 ...

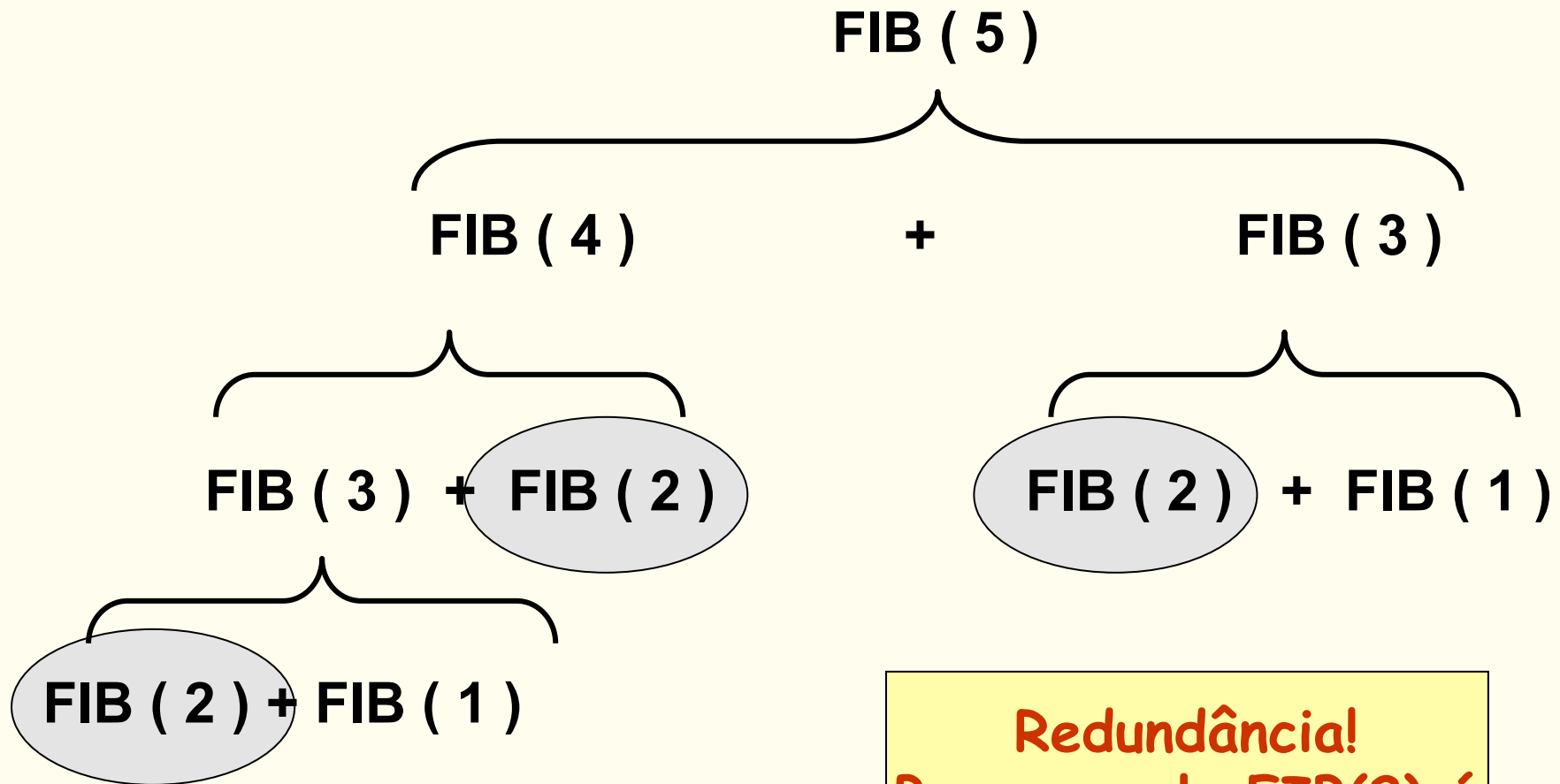
fib (5)?

```
int fib(int n) // solução recursiva
{
    int ret; // retorno da funcao
    /* devolve o termo n da série de fibonacci */
    if (n < 3) // 1º e 2º termo = 1
        ret = 1;
    else
        ret = fib(n-1) + fib(n-2);
    return ret;
}
```

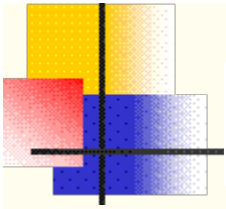
Sequência de Fibonacci



Sequência de Fibonacci



Redundância!
Por exemplo, $FIB(2)$ é
calculado diversas
vezes!!!



Vantagens da Recursividade

- código mais compacto;
- especialmente conveniente para estruturas de dados definidas recursivamente, tais como árvores;
- código pode ser mais fácil de entender.
- implementação mais imediata de funções matemáticas que já são definidas recursivamente.

Desvantagens da Recursividade

- maior ocupação de memória;
- maior tempo de processamento.



Recursividade

→ chamada condicional

(condição de fim da recursão)

```
void rec (x)
{
  ...
  if ...
    return val;
  ...
}
```

→ cada chamada ativa uma rotina na pilha (*stack*)

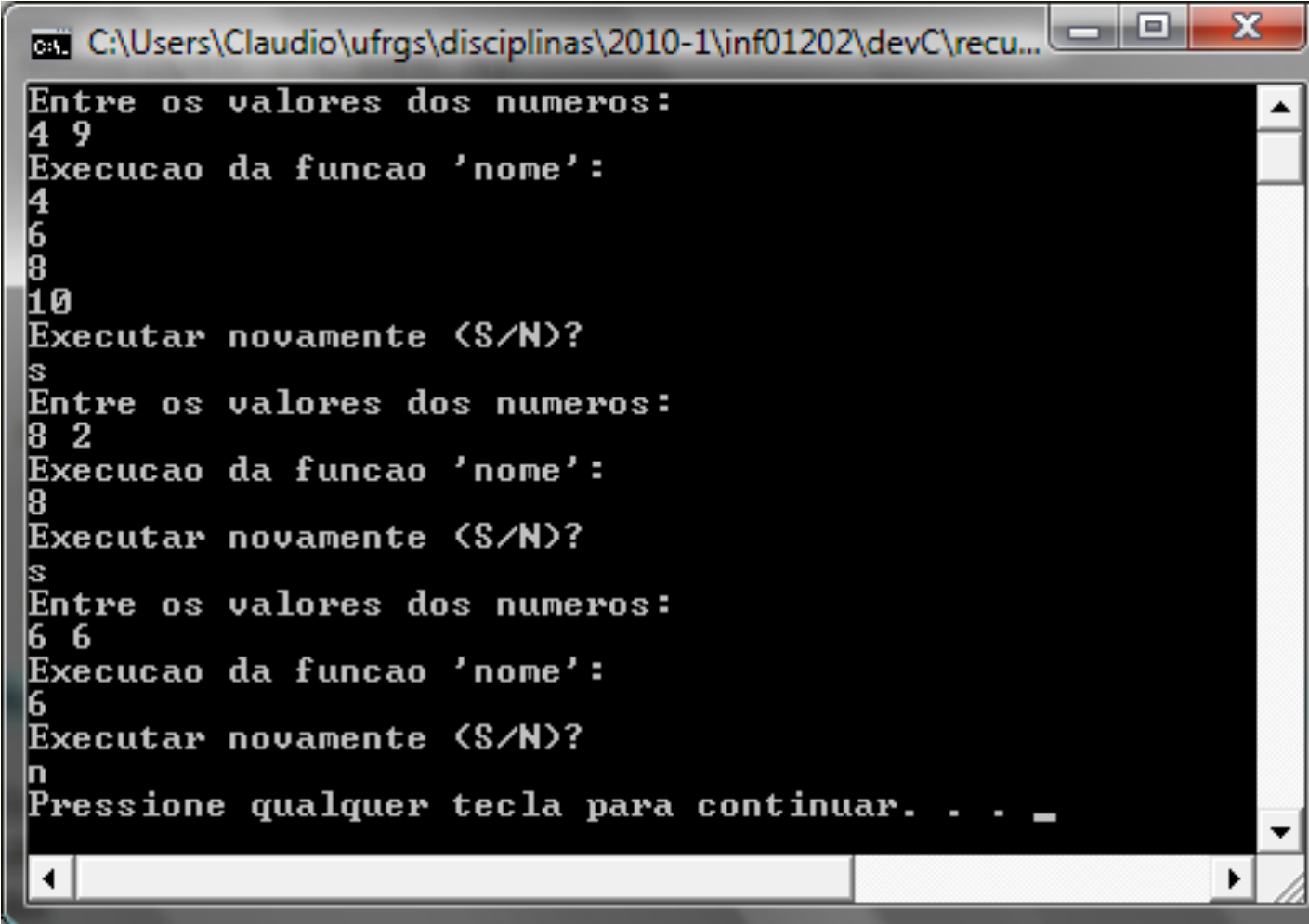
→ sempre pode ser substituído por programação com comandos iterativos

O que faz o programa abaixo?

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void nome(int i, int n)
{
    if (i < n)
    {
        printf("%d\n", i);
        nome(i + 2, n);
    }
    else
    {
        printf("%d\n", i);
    }
}
```

```
int main (void)
{
    int i,n;
    char opc;
    do
    {
        printf("Entre os valores dos numeros:\n");
        scanf("%d%d", &i, &n);
        printf("Execucao da funcao 'nome': \n");
        nome(i,n);
        printf("Executar novamente (S/N)?\n");
        fflush(stdin);
        scanf("%c", &opc);
    } while ( toupper(opc)!='N');
    system("pause");
    return 0;
}
```

O que faz o programa abaixo?



```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf01202\devC\recu...
Entre os valores dos numeros:
4 9
Execucao da funcao 'nome':
4
6
8
10
Executar novamente (S/N)?
s
Entre os valores dos numeros:
8 2
Execucao da funcao 'nome':
8
Executar novamente (S/N)?
s
Entre os valores dos numeros:
6 6
Execucao da funcao 'nome':
6
Executar novamente (S/N)?
n
Pressione qualquer tecla para continuar. . . _
```

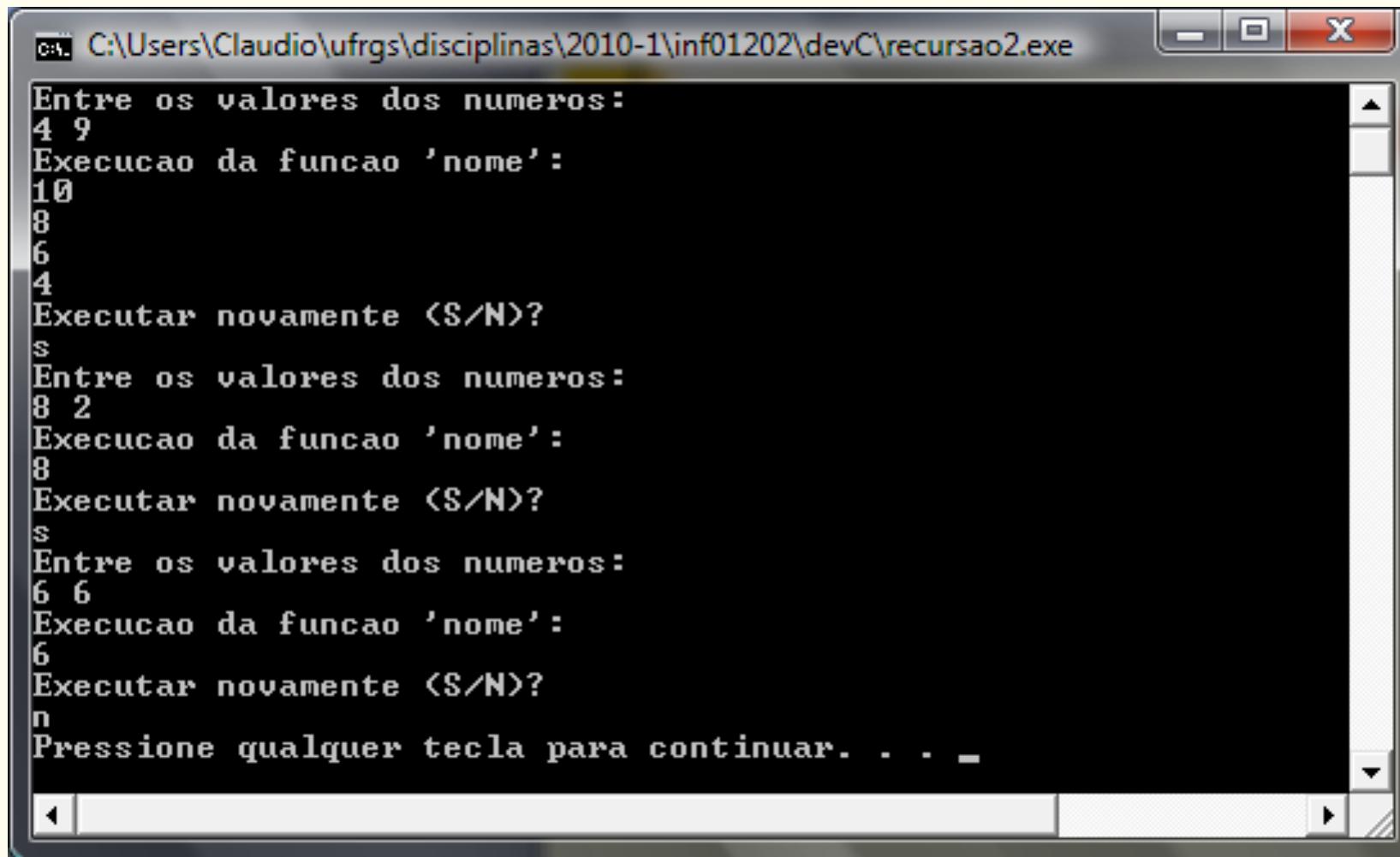
E se a ordem dos comandos abaixo for trocada?

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void nome(int i, int n)
{
    if (i < n)
    {
        nome(i + 2, n);
        printf("%d\n", i);
    }
    else
    {
        printf("%d\n", i);
    }
}
```

```
int main (void)
{
    int i,n;
    char opc;
    do
    {
        printf("Entre os valores dos numeros:\n");
        scanf("%d%d", &i, &n);
        printf("Execucao da funcao 'nome': \n");
        nome(i,n);
        printf("Executar novamente (S/N)?\n");
        fflush(stdin);
        scanf("%c", &opc);
    } while ( toupper(opc)!='N');
    system("pause");
    return 0;
}
```




E se a ordem dos comandos abaixo for trocada?



```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf01202\devC\recursao2.exe
Entre os valores dos numeros:
4 9
Execucao da funcao 'nome':
10
8
6
4
Executar novamente (S/N)?
s
Entre os valores dos numeros:
8 2
Execucao da funcao 'nome':
8
Executar novamente (S/N)?
s
Entre os valores dos numeros:
6 6
Execucao da funcao 'nome':
6
Executar novamente (S/N)?
n
Pressione qualquer tecla para continuar. . . _
```

O que faz a subrotina abaixo?

```
int p ( int x, int y)
{
    int ret; // retorno da funcao
    if ( x == y)
        ret = y;
    else
        ret = y + p ( x , y - 1 );
    return ret;
}
```

$p(1, 4) = ?$

$p(1, 4)$

$4 + p(1, 3)$

$3 + p(1, 2)$

$2 + p(1, 1)$

1

$= 10$

$p(7, 3) = ?$

$p(7, 3)$

$3 + p(7, 2)$

$2 + p(7, 1)$

$1 + p(7, 0)$

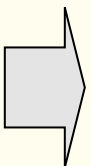
.....

**Cuidado: Recursão
infinita quando $x > y$!!!**

O que faz a subrotina abaixo?

```
void rv( )  
{  
    char ch;  
    scanf("%c", &ch);  
    if (ch != '#')  
        rv();  
    printf("%c", ch);  
}
```

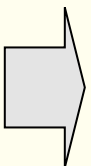
**?
ABC#**

ABC#  **#CBA**

O que faz a subrotina abaixo?

```
void rv( )  
{  
    char ch;  
    scanf("%c", &ch);  
    if (ch != '#')  
        rv();  
    printf("%c", ch);  
}
```

**?
ABC#**

ABC#  **#CBA**

Recursividade mútua

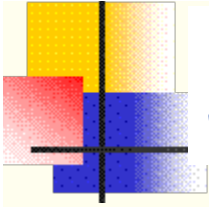
```
int g ( int x); // protótipo, necessário  
pois f definida em função de g.
```

```
int f ( int x)  
{  
    int ret;  
    if (x == 0)  
        ret = 0;  
    else  
        ret = x + g (x);  
    return ret;  
}  
  
int g ( int x)  
{  
    return 4 - f ( x - 1 );  
}
```

resul = f (3); ?

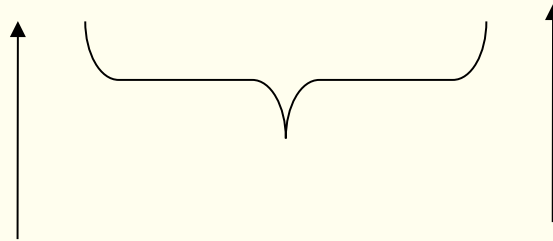
$f(3)$
 $3 + g(3)$
 $4 - f(2)$
 $2 + g(2)$
 $4 - f(1)$
 $1 + g(1)$
 $4 - f(0)$
 0

$3 + 4 - (2 + 4 - (1 + 4 - 0)) \leftrightarrow 6$

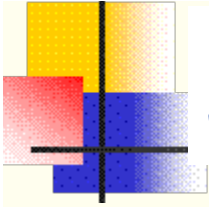


Exercício: escrever uma função recursiva que testa se uma palavra é palíndroma

REVIVER

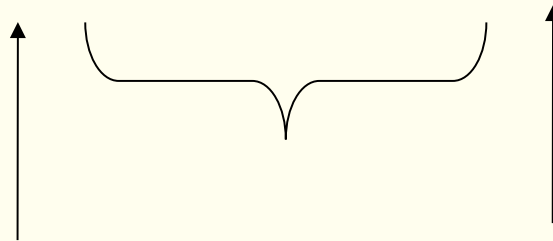


Ideia: comparar a primeira e a última letras. Se forem iguais, testar a substring do meio, recursivamente.

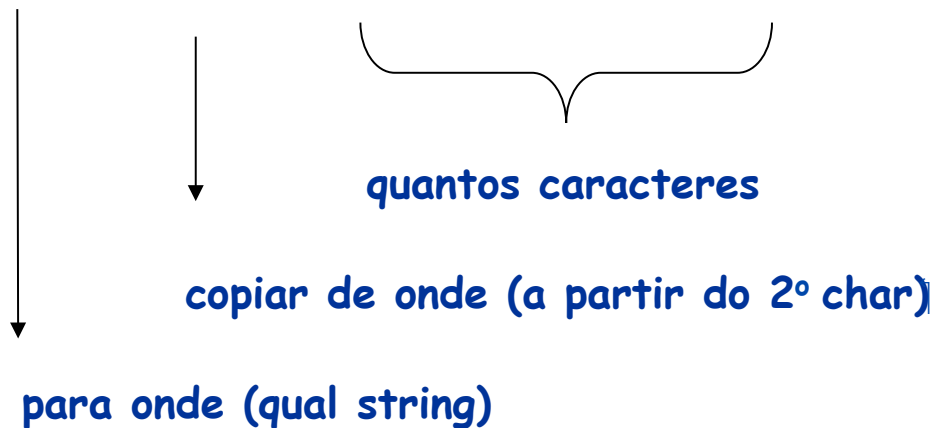


Escrever uma função recursiva que testa se uma palavra é palíndroma

R E V I V E R



```
strncpy (s1, &s[1], strlen(s) - 2);
```





```
/* Programa que determina se uma string lida do teclado é palíndroma ou não.
```

```
Entrada: string lida do teclado
```

```
Saída: mensagem indicando se é palindroma
```

```
*/
```

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

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

```
# include <stdlib.h>
```

```
# define TAMMAX 40
```

```
int palindroma (char*); // prototipo da funcao
```

```
int main (void)
```

```
{
```

```
    char str[TAMMAX];
```

```
    gets(str);
```

```
    if (palindroma(str)==1)
```

```
        printf("%s eh palindroma\n", str);
```

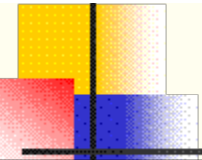
```
    else
```

```
        printf("%s NAO eh palindroma\n", str);
```

```
    system("pause");
```

```
    return 0;
```

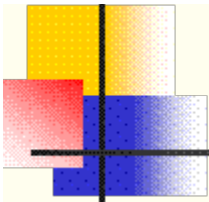
```
}
```

```

int palindroma (char s[ ]) // retorna 1, se a string "s" for palindroma, ou 0, se não for
{
    char s1[TAMMAX]; // define outra string, que recebera a parte central de "s"
    int eh_palind; // saida da funcao
    if (s[0] == s[strlen(s)-1]) // compara a primeira e ultima letras
    {
        if (( strlen(s) == 1 ) || ( strlen(s) == 2)) //se a string tiver 1 ou 2 letras, retorna 1
            eh_palind = 1;
        else
        {
            /* monta uma nova string tirando os 2 extremos */
            strncpy (s1, &s[1], strlen(s) - 2);
            s1[strlen(s) - 2] = '\0'; // coloca o simbolo de final de string
            /* chamada recursiva para a substring central */
            eh_palind = palindroma (s1);
        }
    }
    else // retorna zero se o teste falhar para alguma substring
        eh_palind = 0;
    return eh_palind;
}

```



```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf01202\de...  
REUIVER  
REUIVER eh palindroma  
Pressione qualquer tecla para continuar. . . -
```

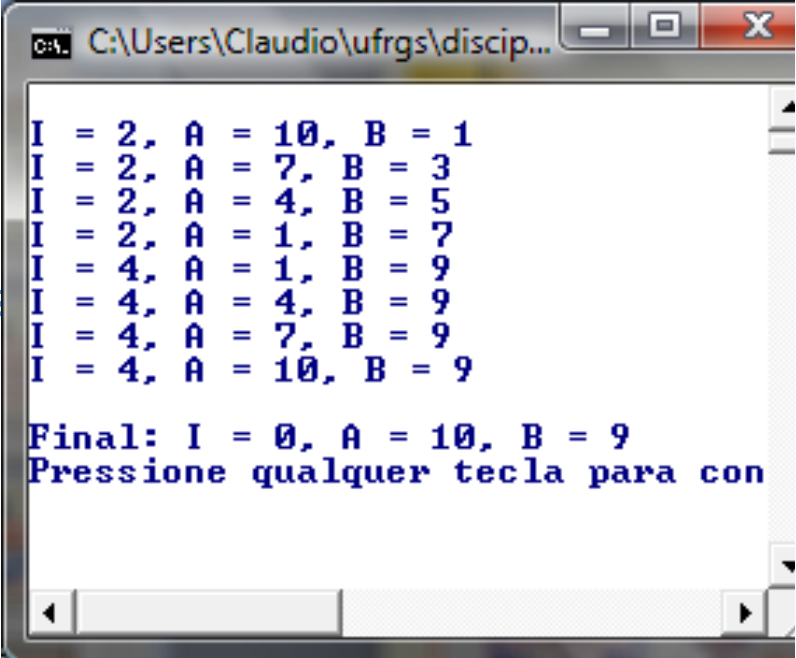
```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf01202\...  
socorram me em marrocos  
socorram me em marrocos eh palindroma  
Pressione qualquer tecla para continuar. . . -
```

```
C:\Users\Claudio\ufrgs\disciplinas\2010-1\inf012...  
abacate azul  
abacate azul NAO eh palindroma  
Pressione qualquer tecla para continuar. . . -
```

Exercício: Qual o resultado da execução do programa abaixo?

```
// Testa recursividade
#include <stdio.h>
#include <stdlib.h>
// Função recursiva:
void x(int a, int *b) // parâmetros: por valor e por referência
{
    int i;
    i = 2;
    printf("\nI = %d, A = %d, B = %d", i, a, *b);
    *b = *b + i; // altera no programa principal!
    if (*b < 8)
        x(a-3, b); // aqui não passa endereço, pois

    i = i + 2; // variável local
    printf("\nI = %d, A = %d, B = %d", i, a, *b);
}
int main()
{
    int i=0, a=10, b=1;
    x(a, &b); // endereço de b
    printf("\n\nFinal: I = %d, A = %d, B = %d", i, a, b);
    printf("\n");
    return 0;
}
```



```
C:\Users\Claudio\ufrgs\discip...
I = 2, A = 10, B = 1
I = 2, A = 7, B = 3
I = 2, A = 4, B = 5
I = 2, A = 1, B = 7
I = 4, A = 1, B = 9
I = 4, A = 4, B = 9
I = 4, A = 7, B = 9
I = 4, A = 10, B = 9

Final: I = 0, A = 10, B = 9
Pressione qualquer tecla para con
```



Exemplos de problemas recursivos

- ordenar arranjo: colocar o menor elemento na primeira posição e ordenar o restante do arranjo
- inverter uma lista: trocar os dois extremos, e inverter o restante da lista
- examinar uma lista: examinar um elemento e examinar resto da lista (até que a lista fique vazia)

Exercício: o método de ordenação por seleção consiste em identificar o menor elemento, colocar este elemento na 1ª posição e repetir o processo para os demais elementos.

Faça o algoritmo e o programa que implemente este método, composto por uma função **recursiva ordena**, uma função **pos_menor** que retorna a posição do menor elemento de um vetor (vetor, posição inicial de busca e posição final de busca devem ser passados como parâmetro) e uma função **troca** que inverte o conteúdo de 2 variáveis passadas como parâmetros. No programa principal, incluir a leitura de um vetor de 10 elementos reais, a chamada da função **ordena** e a impressão do vetor classificado.

Algoritmo Ordena_Selecao;
{ordena vetores pelo método de seleção}
Entrada: vetor;
Saída: vetor classificado em ordem crescente;
1. Inicio
2. Para (ind ← 0; ind < num_elem; ind++)
 lê vetor[ind]
3. Ordena(vetor, pos_inic, pos_fin) **{função}**
4. Para (ind ← 0; ind < num_elem; ind++)
 imprime vetor[ind]
5 Fim.