Recursividade

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema. Esta ferramenta pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema.

Recursividade

- Para se codificar programas de modo recursivo usa-se um procedimento ou sub-rotina, que permite dar um nome a um comando, o qual pode chamar a si próprio.
- Esta chamada pode ser *diretamente recursiva*, quando o procedimento *P* contiver uma referência explícita a si próprio, ou *indiretamente recursiva*, quando o procedimento *P* contiver uma referência a outro procedimento *Q*, que por sua vez contém uma referência direta ou indireta a *P*.

Recursividade

Exemplo: Soma *N* primeiros números inteiros

Supondo *N* = 5;
S(5) = 1+2+3+4+5 = 15
-> S(5) = S(4) + 5 -> 10 + 5 = 15
S(4) = 1+2+3+4 =10
-> S(4) = S(3) + 4 -> 6 + 4 = 10
S(3) = 1+2+3 = 6
-> S(3) = S(2) + 3 -> 3 + 3 = 6
S(2) = 1+2 = 3
-> S(2) = S(1) + 2 -> 1 + 2 = 3
S(1) = 1 =1
-> S(1) = 1 ---------->solução trivial

Recursividade

Em se tratando de procedimentos recursivos pode-se ocorrer um problema de terminação do programa, como um "looping interminável ou infinito".
Portanto, para determinar a terminação das repetições, deve-se:

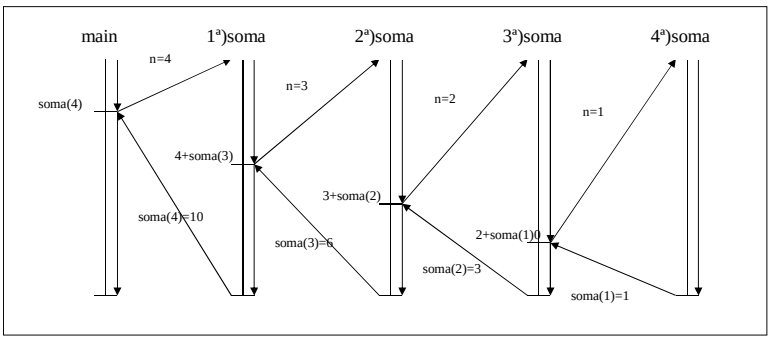
1) Definir uma função que implica em uma condição de terminação (solução trivial), e
2) Provar que a função decresce a cada passo de repetição, permitindo que, eventualmente, esta solução trivial seja atingida.

Recursividade

$$S(N)=\begin{cases}1 & se\ \ N=1 \quad \text{solução trivial}\\ S(N-1)+N, & se\ \ N>1 \quad \text{chamada recursiva}\end{cases}$$

```
main( )
{
  int n;
  scanf("%d", &n);
  printf("%d", soma(n));
}
int soma(int n)
{
  if (n == 1)  return (1);
  else return (n + soma(n – 1));
}
```

Recursividade



Teste de Mesa para N=4 (somatório dos 4 primeiros números inteiros)

Exercício

Determine a ordem de complexidade do algoritmo recursivo de soma de N primeiros números naturais

Recursividade

Seqüência de Fibonacci

Cada termo resulta da adição dos dois que o antecedem ($U_n = U_{n-1} + U_{n-2}$) originando assim os números **0**; **1**; **1**; **2**; **3**; **5**; **8**; **13**; **21** e por aí em diante. Os números desta série têm muitas propriedades e aplicações interessantes, por exemplo na botânica, no desenvolvimento de plantas (arranjo das folhas), e na genética, na produção de coelhos.

Recursividade

Seqüência de Fibonacci

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

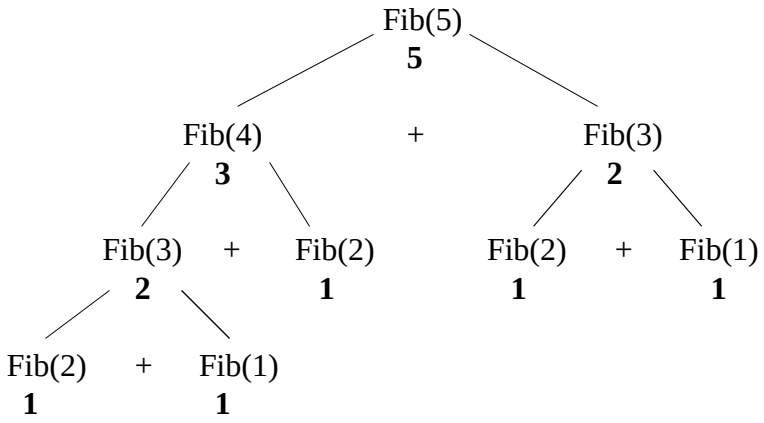
solução trivial

chamada recursiva

```
void main()
{int n;
 scanf("%d", n);
 printf("%d", fibonacci(n));
}
int fibonacci(int n)
{
 if ((n == 1)||(n == 2)) return (1);
 else  return(fibonacci(n-1)+fibonacci(n-2));
}
```

Recursividade

Seqüência de Fibonacci



Recursividade

Seqüência de Fibonacci

Os números de Fibonacci podem ser calculados por um esquema iterativo, que evita o cálculo repetido dos mesmos valores, através do uso de variáveis auxiliares.

```
int cont=1, post=1, ant=0, aux;
while (cont < n)
{
  aux = post;
  post = post + ant;
  ant = aux;
  cont = cont + 1;
}
```

Recursividade

Vantagens X Desvantagens

Um programa recursivo é mais elegante e menor que a sua versão iterativa, além de exibir com maior clareza o processo utilizado, desde que o problema ou os dados sejam naturalmente definidos através de recorrência.

Por outro lado, um programa recursivo exige mais espaço de memória e é, na grande maioria dos casos, mais lento do que a versão iterativa.

Recursividade

Fatorial

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

Recursividade

Fatorial

factorial(20) -- allocate 1 word of memory,
call factorial(19) -- allocate 1 word of memory,
call factorial(18) -- allocate 1 word of memory,
...
call factorial(2) -- allocate 1 word of memory,
call factorial(1) -- allocate 1 word of memory,
call factorial(0) -- allocate 1 word of memory,

at this point 21 words of memory and 21 activation records have been allocated.

return 1. -- release 1 word of memory.
*return 1*1. -- release 1 word of memory.*
*return 2*1. -- release 1 word of memory.*
...

Recursividade

Torre de Hanoi

Existem três estacas A, B e C. Vários discos de diferentes diâmetros são encaixados na estaca A, de modo que um disco maior fique sempre abaixo de um disco menor, O objetivo é deslocar os discos para a estaca C, usando a estaca B como auxiliar. Somente o primeiro disco de toda estaca pode ser deslocado para outra estaca, e um disco maior não pode nunca ficar posicionado sobre um disco menor.

Recursividade

Torre de Hanoi

```
#include <stdio.h>
main()
{
    int n;
    scanf("%d", &n);
    towers(n, 'A', 'B', 'C');
}
```

```
towers(int n, char frompeg, char topeg, char auxpeg)
{
    if(n == 1){
        printf("\n%s%c%s%c", "move disco 1 da estaca",
                                frompeg, "p/ a estaca", to peg);
        return;
     }
     towers(n-1, frompeg, auxpeg, topeg);
     printf("\n%s%d%s%c%s%c", "move disco", n, "da estaca",
                                    frompeg, "p/ a estaca", to peg);
     towers(n-1, auxpeg, topeg, frompeg);
}
```

Recursividade

Exercício

Determine a ordem de complexidade do algoritmo recursivo de Torre de Hanoi