



Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Recursão em C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010

ROTEIRO DA AULA

- Definição de recursão
- Exemplos de recursão
- Estrutura da recursão
- Vantagens e desvantagens da recursão

RECURSÃO EM C

uma função é dita **recursiva** quando dentro do seu código existe **uma chamada para si mesma**

Ex: cálculo do fatorial de um número:

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

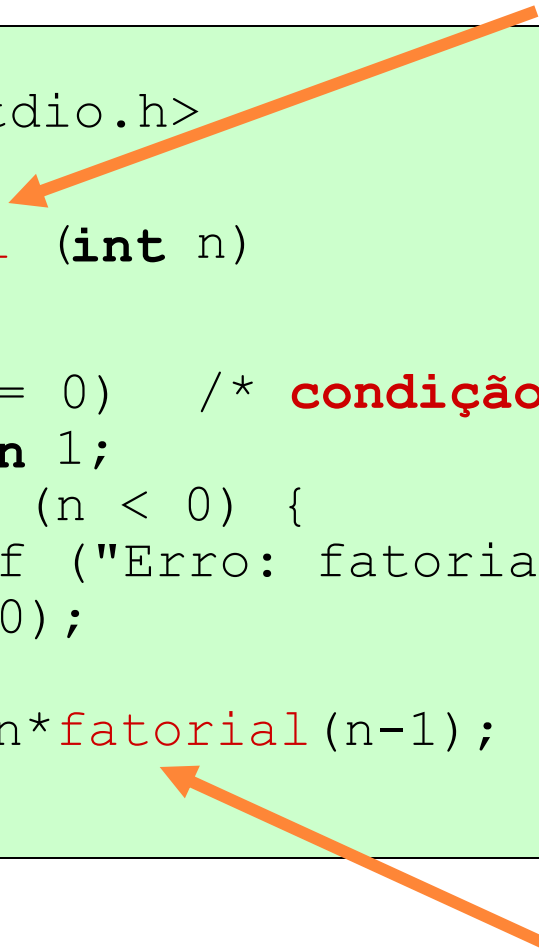
RECURSÃO EM C

- 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

EXEMPLO FATORIAL

```
#include <stdio.h>

int fatorial (int n)
{
    if (n == 0)    /* condição de parada da recursão */
        return 1;
    else if (n < 0) {
        printf ("Erro: fatorial de número negativo!\n");
        exit(0);
    }
    return n*fatorial(n-1);
}
```



EXEMPLO *FATORIAL*

```
fatorial(5)
=> (5 ° 0)
  return 5 • fatorial(4)
=> (4 ° 0)
  return 4 • fatorial(3)
=> (3 ° 0)
  return 3 • fatorial(2)
=> (2 ° 0)
  return 2 • fatorial(1)
=> (1 ° 0)
  return 1 • fatorial(0)
=> (0 == 0)
  <= return 1
  <= return 1 • 1      (1)
  <= return 2 • 1      (2)
  <= return 3 • 2      (6)
  <= return 4 • 6      (24)
<= return 5 • 24      (120)
```

120

```
/* Faça um programa que calcula e mostra o fatorial de  
um número inteiro positivo n. (Ex: 0! = 1; 3! = 3*2*1 = 6)  
*/
```

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

```
int fatorial (int n)  
{  
    if (n == 0)  
        return 1;  
    else if (n<0){  
        printf("\nErro: fatorial de numero negativo!\n");  
        getch();  
        exit(0);  
    }  
    return n*fatorial(n-1);  
}
```

```
int main(void)
{
int n, fat=1;
    printf("Digite um numero inteiro:");
    scanf("%d", &n);
    fat=fatorial(n);
    printf("\n\nO fatorial de %d eh: %d", n,
fat);
    getch();
}
```


EX: SOMA N PRIMEIROS NÚMEROS INTEIROS

Supondo $N = 5$;

$$S(5) = 1+2+3+4+5 = 15$$

$$\rightarrow S(5) = S(4) + 5 \rightarrow 10 + 5 = 15$$

$$S(4) = 1+2+3+4 = 10$$

$$\rightarrow S(4) = S(3) + 4 \rightarrow 6 + 4 = 10$$

$$S(3) = 1+2+3 = 6$$

$$\rightarrow S(3) = S(2) + 3 \rightarrow 3 + 3 = 6$$

$$S(2) = 1+2 = 3$$

$$\rightarrow S(2) = S(1) + 2 \rightarrow 1 + 2 = 3$$

$$S(1) = 1 = 1$$

$$\rightarrow S(1) = 1 \text{ -----} \rightarrow \text{solução trivial}$$

RECURSÃO

- Em procedimentos recursivos pode 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.

ESTRUTURA DE UMA RECURSÃO

- uma recursão obedece a uma estrutura que deve conter os seguintes elementos:

função (par)

- teste de término de recursão utilizando **par**
 - *se teste ok, retorna aqui*
- processamento
 - *aqui a função processa as informações em **par***
- chamada recursiva em **par'**
 - *par deve ser modificado de forma que a recursão chegue a um término*

EX: SOMA N PRIMEIROS NÚMEROS INTEIROS

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

```
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));
}
```

EXEMPLO: *PRINTF(INT)*

- *printf(int)* imprime um inteiro usando recursão

```
void printf (int n) {  
  if (n < 0) {          /* imprime sinal */  
    putchar('-');  
    n = -n;  
  }  
  if (n / 10)          /* termino recursao */  
    printf(n/10); /* recursao se n>10 */  
  putchar(n % 10 + '0'); /* senao imprime char */  
}
```

EXEMPLO *PRINTF*()

```
printf(-1974)
```

```
==> (n < 0) --> putchar('-')
```

```
==> printf(197)
```

```
==> printf(19)
```

```
==> printf(1)
```

```
(1 / 10 == 0)
```

```
putchar(1 + '0')
```

```
putchar(19%10 + '0')
```

```
putchar(197%10 + '0')
```

```
putchar(1974 %10 + '0')
```

-

-

-

-

-

-1

-19

-197

-1974

ANALISANDO 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.

PRÁTICA

1. Escreva uma função recursiva para calcular o valor de uma base x elevada a um expoente y .
2. Escrever uma função recursiva que retorna o tamanho de um *string*, `tamstring(char s[])`
3. Fazer uma função recursiva que conta o número de ocorrências de um determinado caracter, `caract(char c, char s[])`
4. Escreva uma função recursiva que produza o reverso de um *string*, `reverse(char s[])`


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

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

```
int main(void) {
int x, y, e;
    printf("Exponencial de x elevado a y\n\n");
    printf("\nDigite o numero inteiro x:");
    scanf("%d", &x);
    printf("\nDigite o numero inteiro y:");
    scanf("%d", &y);
    if (y < 0) {
        printf("y tem que ser maior ou igual a zero!!");
        getch(); }
    else{
        e=expo(x,y);
        printf("\n\nX elevado a y eh: %d", e);
        getch();}
}
```

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

```
int tamstring(char s[])
{
    if (s[0] == '\0')
        return 0;
    return 1+tamstring(&s[1]);
}
```

```
int main(void)
{
    char s[20];
    int t;
    printf("Tamanho de string\n\n");
    printf("\nDigite a string: ");
    scanf("%s", s);
    t=tamstring(s);
    printf("\n\nO tamanho eh %d", t);
    getch();
}
```

```
/* conta quantas vezes um caractere ocorre em uma string*/
```

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

```
#include <conio.h>
```

```
int carac(char c,char s[])
```

```
{
```

```
    if (s[0] == '\0')
```

```
        return 0;
```

```
    if (s[0]==c) return (1+carac(c,++s));
```

```
    return carac(c,++s);
```

```
}
```

```
int main(void)
```

```
{
```

```
    char s[30],c;
```

```
    int t;
```

```
    printf("Busca em string\n\n");
```

```
    printf("\nDigite a string: ");
```

```
    gets(s);
```

```
    printf("\nDigite o caractere desejado: ");
```

```
    c=getchar();
```

```
    t=carac(c,s);
```

```
    printf("\n\nEncontrei %d vezes", t);
```

```
    getch();
```

```
}
```

```
/* imprime uma string em ordem reversa*/
```

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

```
#include <conio.h>
```

```
void contrario(char s[])
```

```
{
```

```
    if (s[0] != '\0'){
```

```
        contrario(&s[1]);
```

```
        printf("%c",s[0]);}
```

```
}
```

```
int main(void)
```

```
{
```

```
    char s[30],c;
```

```
    int t;
```

```
    printf("Imprime reverso\n\n");
```

```
    printf("\nDigite a string: ");
```

```
    gets(s);
```

```
    contrario(s);
```

```
    getch();
```

```
}
```



Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Recursão em C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010