Funções

Roberto Rocha

PUC Minas Virtual

E se uma função chamar a si mesma?

Funções recursivas

→ Funções podem ser chamadas recursivamente, dentro do corpo de uma função podemos chamar novamente a própria função.

→ As implementações recursivas devem ser pensadas conforme a definição recursiva do problema que se deseja resolver.

→ Exemplo: Valor do fatorial de um número definido de forma recursiva:

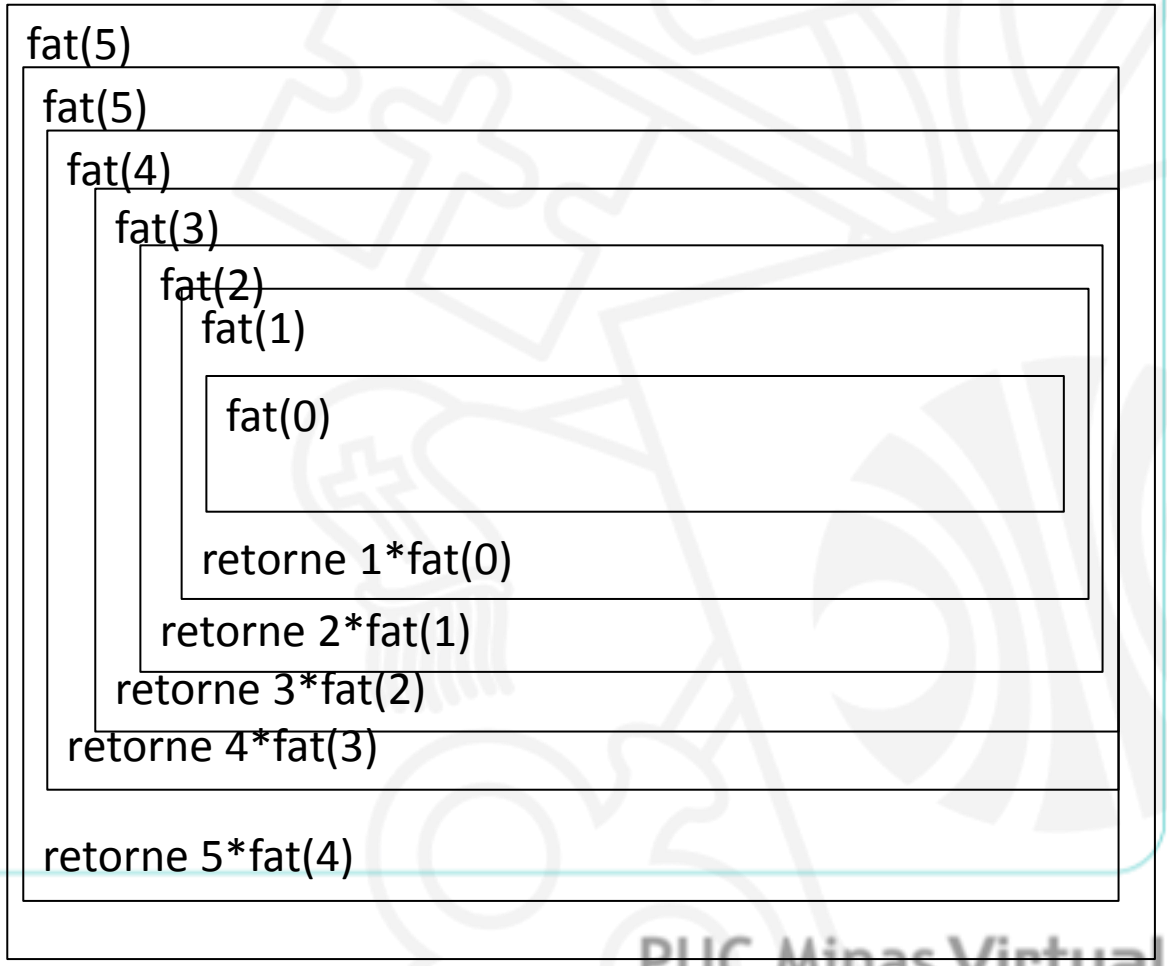
$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n \times (n-1)!, \text{ se } n > 0 \end{cases}$$

Funções recursivas

$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n \times (n-1)!, \text{ se } n>0 \end{cases}$$

```
função fat ( n:inteiro):inteiro
inicio
    se (n=0) então
        retorne 1
    senão
        retorne n*fat(n-1)
fimfunção

inicio
  escreva(fat(5))
fimalgoritmo
```



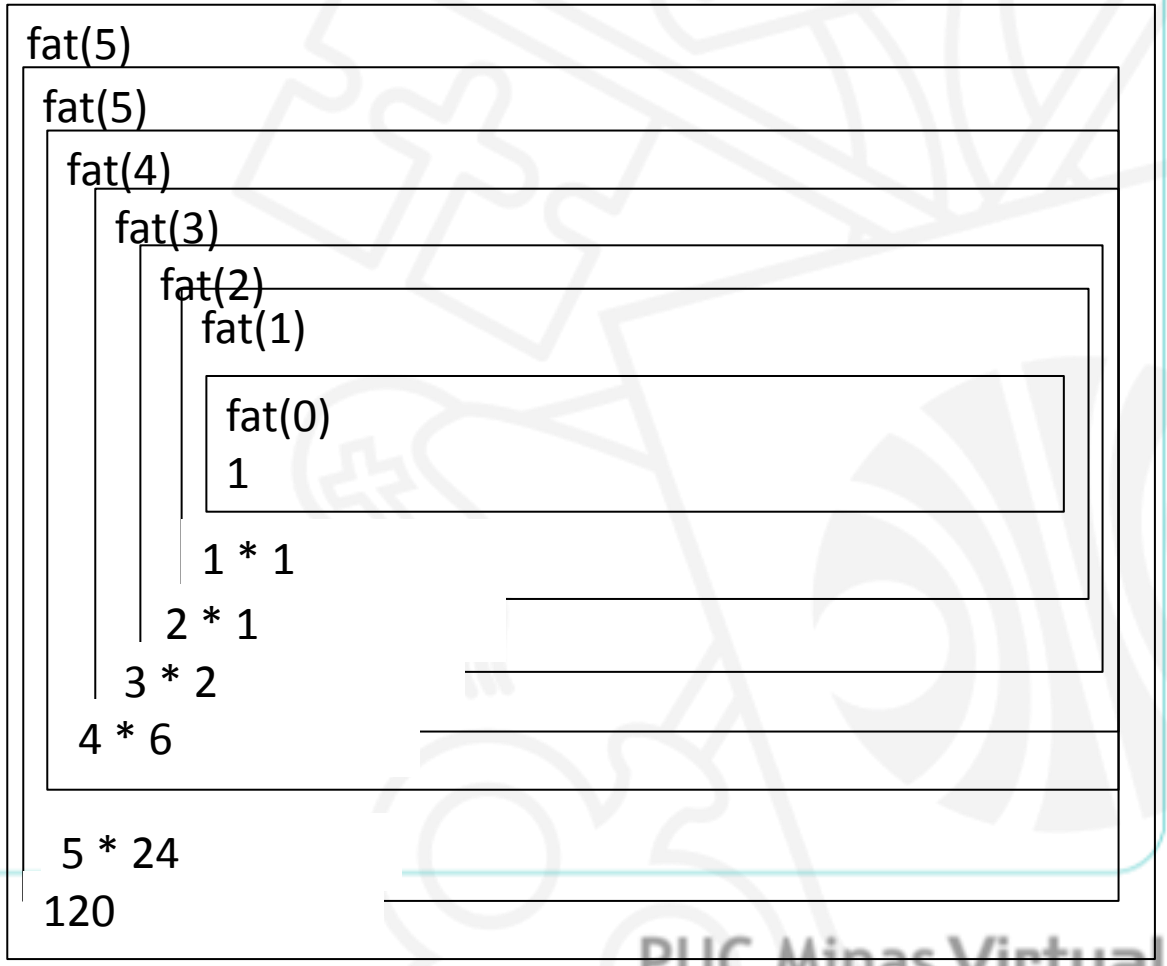
Funções recursivas

$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n \times (n-1)!, \text{ se } n > 0 \end{cases}$$

```
função fat ( n:inteiro):inteiro
inicio
    se (n=0) então
        retorne 1
    senão
        retorne n*fat(n-1)
fimfunção

inicio
  escreva(fat(5))
fimalgoritmo
```

120



Funções recursivas

```
função fat ( n:inteiro):inteiro
inicio
   se (n=0) então
      retorne 1
   senão
      retorne n*fat(n-1)
fimfunção

inicio
  escreva(fat(5))
fimalgoritmo
```

```c
#include <stdio.h>
#include <stdlib.h>
int fat(int n);
int main()
{
    printf("fatorial de 5=%d\n",fat(5));
    return 0;
}

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

```
fatorial de 5=120

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Faça **um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário introduziu ou seja: 1 + 2 + 3 + ... + n**

a) faça sem utilizar recursividade

b) utilize agora recursividade.

Funções recursivas

Exercício de fixação:

Faça **um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário introduziu ou seja: 1 + 2 + 3 + ... + n**

a) faça sem utilizar recursividade

b) utilize agora recursividade.

```
Algoritmo “Soma1aN”
var
soma,i,n:inteiro
inicio
   leia(n)
   s←0
   para i de 1 ate n passo 1 faça
     s←s+i
   fimpara
   escreva(s)
fimalgoritmo
```

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

int main()
{
    int soma,i,n;
    printf("Digite um valor:");
    scanf("%d",&n);
    soma=0;
    for(i=1;i<=n;i=i+1){
        soma=soma+i;
    }
    printf("A soma dos numero de 1 a %d = %d\n",n,soma);
    return 0;
}
```

```
Digite um valor:5
A soma dos numero de 1 a 5 = 15

Process returned 0 (0x0)   execution time : 20.843 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Faça **um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário introduziu ou seja: 1 + 2 + 3 + ... + n**

a) faça sem utilizar recursividade

b) utilize agora recursividade.

Vamos criar uma definição para a solução

$$\sum_{1}^{n} \begin{cases} \textbf{1, se n = 1} \\ n + \sum_{1}^{n-1} & \textbf{se n>1} \end{cases}$$

```
função soma (n:inteiro):inteiro
inicio
    se (n=1) então
        retorne 1
    senão
        retorne n+soma(n-1)
fimfunção

 inicio
   escreva(fat(5))
 fimalgoritmo
```

Funções recursivas

Exercício de fixação:

Faça **um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário introduziu ou seja: 1 + 2 + 3 + ... + n**

a) faça sem utilizar recursividade

b) utilize agora recursividade.

```
função soma (n:inteiro):inteiro
inicio
    se (n=1) então
        retorne 1
    senão
        retorne n+soma(n-1)
fimfunção

inicio
  escreva(fat(5))
fimalgoritmo
```

```c
#include <stdio.h>
#include <stdlib.h>
int soma(int n);
int main()
{
    printf("A soma de 1 ate %d = %d\n",5,soma(5));
    return 0;
}

int soma(int n)
{
    if (n==1)
        return 1;
    else
        return n+soma(n-1);
}
```

```
A soma de 1 ate 5 = 15

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Crie uma função que receba 2 parâmetros(x,y) e devolva x elevado a y.

a) faça sem utilizar recursividade

b) utilize agora recursividade.

Funções recursivas

Exercício de fixação:

Crie uma função que receba 2 parâmetros(x,y) e devolva x elevado a y.

a) faça sem utilizar recursividade

b) utilize agora recursividade.

```
função potencia (x,y:inteiro):inteiro
var i,pot:inteiro
inicio
    pot ← 1
    para i de 1 ate y passo 1 faça
      pot ← pot * x
    fimpara
    retorne pot
fimfunção
inicio
  escreva(pot(2,3))
fimalgoritmo
```

```c
#include <stdio.h>
#include <stdlib.h>
int potencia(int x, int y);
int main()
{
    printf("A resposta de 2 elevado a 3 = %d!\n",potencia(2,3));
    return 0;
}

int potencia(int x, int y)
{
    int i,pot;
    pot=1;
    for (i=1;i<=y;i=i+1)
    {
        pot=pot*x;
    }
    return pot;
}
```

```
A resposta de 2 elevado a 3 = 8!

Process returned 0 (0x0)   execution time : 3.133 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Crie uma função que receba 2 parâmetros(x,y) e devolva x elevado a y.

a) faça sem utilizar recursividade

b) utilize agora recursividade.

$$x^y \begin{cases} 1, \text{ se } y = 0 \\ x * x^{y-1} \end{cases}$$

```
função potencia (x,y:inteiro):inteiro
inicio
    se (y=0) então
      retorne 1
    senão
      retorne x * potencia(x,y-1)
     fimse
fimfunção

inicio
  escreva(pot(2,3))
fimalgoritmo
```

Funções recursivas

Exercício de fixação:

Crie uma função que receba 2 parâmetros(x,y) e devolva x elevado a y.

a) faça sem utilizar recursividade

b) utilize agora recursividade.

```
função potencia (x,y:inteiro):inteiro
inicio
    se (y=0) então
      retorne 1
    senão
      retorne x * potencia(x,y-1)
     fimse
fimfunção

inicio
  escreva(pot(2,3))
fimalgoritmo
```

```c
#include <stdio.h>
#include <stdlib.h>
int potencia(int x, int y);
int main()
{
    printf("A resposta de 2 elevado a 3 = %d!\n",potencia(2,3));
    return 0;
}

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

```
A resposta de 2 elevado a 3 = 8!

Process returned 0 (0x0)   execution time : 3.397 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Crie uma função que imprima o enésimo termo da serie de Fibonacci

a) faça sem utilizar recursividade

b) utilize agora recursividade.

Série de Fibonacci

Entrada

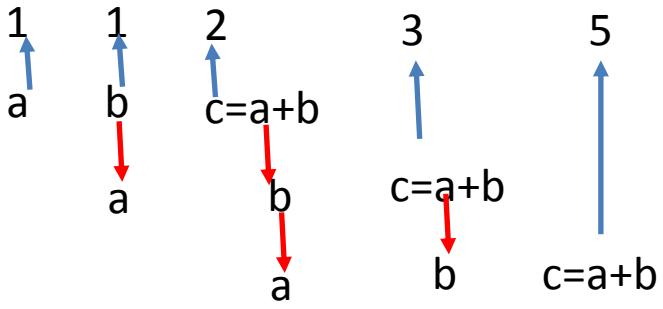
Número

Processamento

Imprimir N números da série de Fibonacci

Saída

Imprimir o enésimo termo da série de Fibonacci



a=b
b=c

Imprimir os N primeiros termos da Série de Fibonacci

```
Algoritmo “Fibonacci”
// Imprimir os N primeiros termos da Série de Fibonacci
var
  numero, a,b,c,cont:inteiro
inicio
   leia(numero)
    a ←1
    b ←1
    para cont de 2 ate numero-1) faca
        c ←a+b
        a ←b
        b ←c
  fimpara
 escreva(c)
fimalgoritmo
```

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

int main()
{
    int a,b,c,cont,numero;
    printf("Digite o termo desejado:");
    scanf("%d",&numero);
    a=1;
    b=1;
    for (cont=2;cont<numero;cont=cont+1)
    {
        c=a+b;
        a=b;
        b=c;
    }
    printf("O %d termo da serie = %d\n",numero,c);
    return 0;
}
```

```
Digite o termo desejado:6
O 6 termo da serie = 8

Process returned 0 (0x0)   execution time : 2.159 s
Press any key to continue.
```

Funções recursivas

Exercício de fixação:

Crie uma função que imprima o enésimo termo da serie de Fibonacci

a) faça sem utilizar recursividade

b) utilize agora recursividade.

$$F(n) \begin{cases} 1, \text{ se } n = 1 \\ 1, \text{ se } n = 2 \\ f(n-1)+f(n-2), \text{ se } n>2 \end{cases}$$

```
função fibonacci(n:inteiro):inteiro
inicio
    se (n=1 ou n = 2) então
      retorne 1
    senão
      retorne fibonacci(n-1) + fibonacci(n-2)
     fimse
fimfunção


inicio
  escreva(fibonacci(6))
fimalgoritmo
```

Funções recursivas

Exercício de fixação:

Crie uma função que imprima o enésimo termo da serie de Fibonacci

a) faça sem utilizar recursividade

b) utilize agora recursividade.

```
função fibonacci(n:inteiro):inteiro
inicio
    se (n=1 ou n = 2) então
      retorne 1
    senão
      retorne fibonacci(n-1) + fibonacci(n-2)
     fimse
fimfunção

inicio
  escreva(fibonacci(6))
fimalgoritmo
```

```c
#include <stdio.h>
#include <stdlib.h>
int fibonacci(int n);
int main()
{
    printf("O %d termo da serie = %d\n",6,fibonacci(6));
    return 0;
}
int fibonacci(int n)
{
    if (n==1 || n==2)
    {
        return 1;
    }
    else
    {
        return fibonacci(n-1)+fibonacci(n-2);
    }
}
```

```
O 6 termo da serie = 8

Process returned 0 (0x0)   execution time : 3.355 s
Press any key to continue.
```

PUC Minas
Virtual