AED1 ou LTP2

Recursividade

Profa.: Márcia Sampaio Lima

EST - UEA

Recursividade

- É uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema.

- Técnica utilizada sempre que for possível expressar a solução de um problema em função do próprio problema.

Recursividade

- Uma função recursiva é aquela que invoca ela mesma.
  - É normal que uma função invoque outra.
- Porém, é possível que uma função chame ela mesma!!!

```
int fat(int i){
   ....
   fat(5);
   ....

}
```

Recursividade

- É preciso um cuidado especial para não cairmos em um *loop* infinito.
  - Um problema que surge ao usar a recursividade é como fazê-la parar.
  - É fácil cair num loop infinito recursivo o qual pode inclusive esgotar a memória.

Recursividade

- **Como funciona a recursividade?**
  - Em uma função recursiva, a cada chamada é criada na memória uma nova ocorrência da função com comandos e variáveis “isolados” das ocorrências anteriores.
  - A função é executada até que todas as ocorrências tenham sido resolvidas.

Recursividade

- **Vantagens da recursividade**
  - Torna a escrita do código mais simples e elegante, tornando-o fácil de entender e de manter.

Recursividade

- **Desvantagens da recursividade**
  - Quando o loop recursivo é muito grande é consumida muita memória nas chamadas a diversos níveis de recursão, pois cada chamada recursiva aloca memória para os parâmetros, variáveis locais e de controle.
  - Em muitos casos uma solução iterativa gasta menos memória, e torna-se mais eficiente do que usar recursão.

Recursividade

- Toda recursividade é composta por:
  - um **caso base** e
  - pelas **chamadas recursivas,**.
- **Caso base:** é o caso mais simples. É usada uma condição em que se resolve o problema com facilidade.
- **Chamadas Recursivas:** procuram simplificar o problema de tal forma que convergem para o caso base.

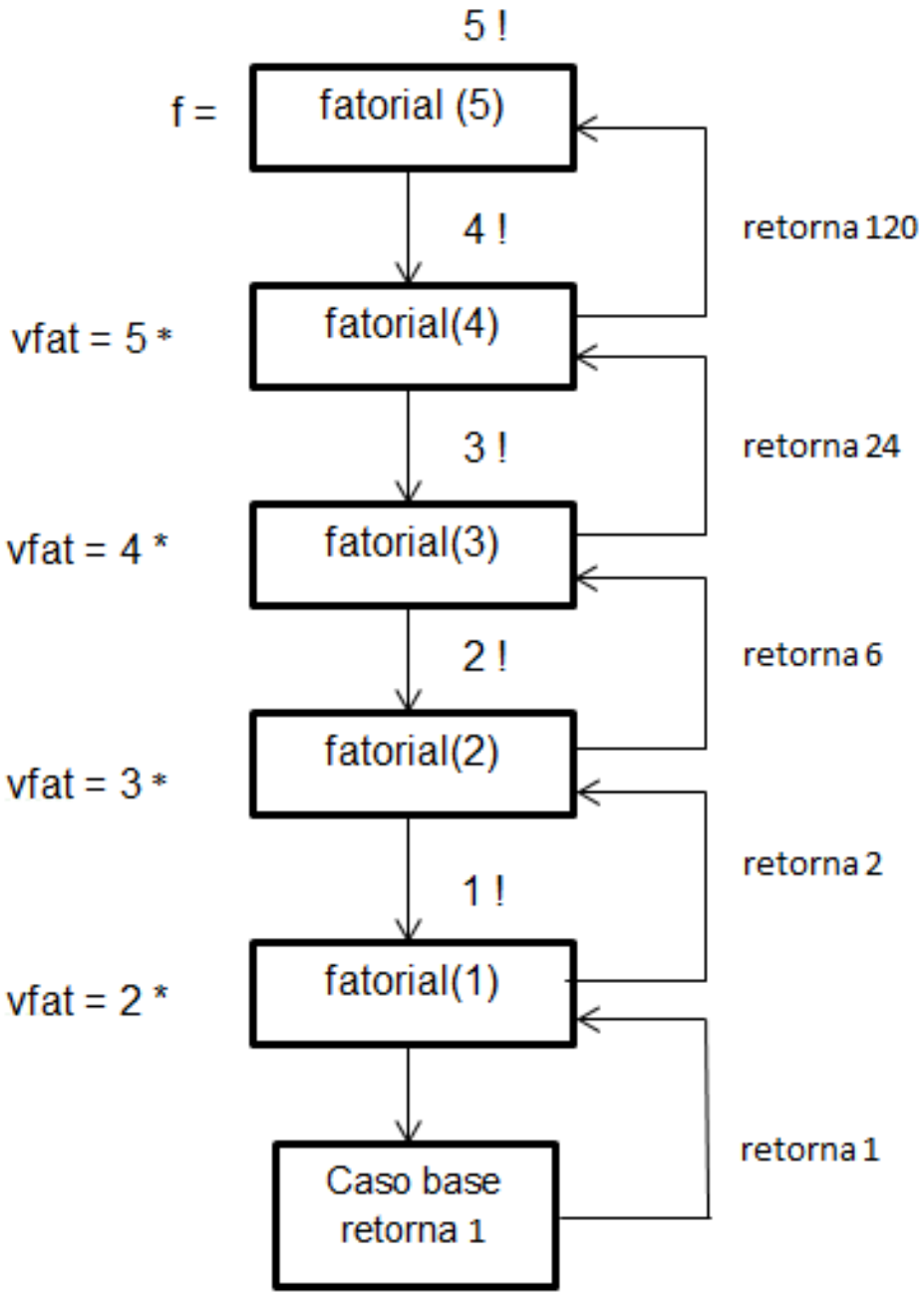
Recursividade

```
//Função recursiva que calcula o fatorial
//de um numero inteiro n
int fatorial(int n)
{

   int vfat;

  if ( n <= 1 )
     //Caso base: fatorial de n <= 1 retorna 1
     return (1);
  else
  {
     //Chamada recursiva
     vfat = n * fatorial(n - 1);
     return (vfat);
  }
}
```

Recursividade

- No programa, se **n <=** 1 então o valor 1 será retornado e a função encerrada (sem necessidade de chamadas recursivas).
- Caso contrário dá-se início a chamadas recursivas até cair no caso mais simples (**n <=** 1) :
  - as chamadas retornam valores de forma a solucionar o cálculo.

Recursividade



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

//protótipo da função fatorial
double fatorial(int n);

int main(void)
{
  int numero;
  double f;

  printf("Digite o numero que deseja calcular o fatorial:
");
  scanf("%d",&numero);

  //chamada da função fatorial
  f = fatorial(numero);

  printf("Fatorial de %d = %.0lf",numero,f);

  getch();
  return 0;
}
```

```
//Função recursiva que calcula o fatorial
//de um numero inteiro n
int fatorial(int n)
{

   int vfat;

  if ( n <= 1 )
     //Caso base: fatorial de n <= 1 retorna 1
     return (1);
  else
  {
     //Chamada recursiva
     vfat = n * fatorial(n - 1);
     return (vfat);
  }
}
```

Recursividade

- Estrutura de uma função recursiva:
- função (valor)
  - Teste de término de recursão utilizando valor
    - se teste ok, retorna aqui
  - Processamento
    - aqui a função processa as informações em valor
  - Chamada recursiva em valor'
    - Valor deve ser modificado de forma que a recursão chegue a um término

Recursividade

- **Crie um programa em C 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**
- **Utilize recursividade.**

Recursividade

- Criar uma função *soma(int n)*.
  - Se n=5, essa função deve retornar:
    - soma(5) = 5 + 4 + 3 + 2 + 1
  - Se n=4, essa função deve retornar:
    - soma(4) = 4 + 3 + 2 + 1
  - Se n=3, essa função deve retornar:
    - soma(3) = 3 + 2 + 1
  - .....

Recursividade

- Para fazermos uso da recursividade, temos que notar padrões.
  - soma(5) = 5 + 4 + 3 + 2 +1 = 5 + soma(4)
  - soma(4) = 4 + 3 + 2 + 1 = 4 + soma(3)
  - soma(3) = 3 + 2 + 1 = 3 + soma(2)
  - soma(2) = 2 + 1 = 2 + soma(1)
  - soma(1) = 1 = 1

  - A fórmula geral:
  - soma(n) = n + soma(n-1)

Recursividade

- Ou seja:
  - soma(n) = n + soma(n-1)
    = n + (n-1) + soma(n-2)
    = n + (n-1) + (n-2) + soma(n-3)
    ...
- E onde essa soma para?
  - quando o último elemento dessa soma for 1.
- Então:
  - soma(n) = n +(n-1) + (n-2) + (n-3) + ... + 1

Recursividade

- Traduzir a fórmula em termos de programação.
  - A função recebe um número, e a primeira coisa a ser feita é testar se esse valor é 1.
  - Se for, deve retornar 1, afinal:
    - soma(1) = 1
  - E se não for 1, deve retornar:
    - n + soma(n-1)

```c
#include <stdio.h>

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

int main()
{
    int n;
    printf("Digite um inteiro positivo: ");
    scanf("%d", &n);
    printf("Soma: %d\n", soma(n));
}
```

Recursividade

- Escreva uma função recursiva para calcular o valor de uma base x elevada a um expoente y.

```
#include stdio.h
int expo (int x, int y)
  {
        if (y == 0)
            {return 1;}
         else
           { return (x*expo(x,y -1)); }
}
int main (void)
{
      int x, y, e;
      printf(" \nDigite o valor da base :");
      scanf("%d", &x);
      printf(" \nDigite o valor do expoente:");
      scanf("%d", &y);
      if (y < 0) {
           printf("y tem que ser maior ou igual a zero!!");
      }
      else {
            e=expo(x,y);
            printf(" \ n \nX elevado a y eh: %d", e);
      } }
```

Recursividade

- Escrever uma função recursiva que retorna o tamanho de um string, tamstring(char s[]).
  - Sabendo que:
    - Uma String é um vetor de char;
    - Caractere que finaliza uma String é ‘\0’ ou 0.

```
#include<stdio.h>

int tamStr(char texto[]);

main(){
	char nome[30];
	int tam;

	printf("Informe seu nome:");
	gets (nome);

	tam = tamStr(nome);

	printf("Resultado: %d", tam);

}
```

```
int tamStr(char texto[]){

      if (texto[0]=='\0'){
            return 0;
      }else{
            return ( 1 + tamStr(&texto[1]));
      }

}
```