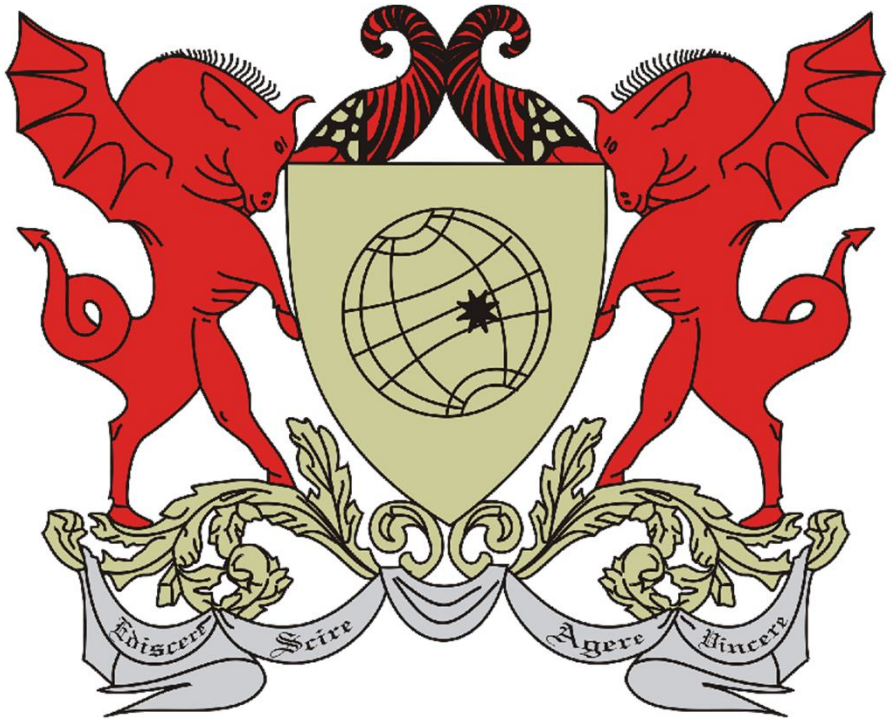

CCF 110 – Programação

Aula 09 – Funções Recursivas

Prof. José Augusto Nacif – jnacif@ufv.br

Conceito de Recursividade

- Fundamental em Matemática e Ciência da Computação
  - Um programa recursivo é um programa que chama a si mesmo
  - Uma função recursiva é definida em termos dela mesma
- Exemplos
  - Números naturais, Função fatorial, Árvore
- Conceito poderoso
  - Define conjuntos infinitos com *comandos* finitos

Recursividade

- A recursividade é uma estratégia que pode ser utilizada sempre que o cálculo de uma função para o valor n, pode ser descrita a partir do cálculo desta mesma função para o termo anterior (n-1).

Exemplo – Função fatorial:

n! = n * (n-1) * (n-2) * (n-3) *....* 1
(n-1)! = (n-1) * (n-2) * (n-3) *....* 1

logo:

n! = n * (n-1)!

Recursividade

- Definição: dentro do corpo de uma função, chamar novamente a própria função
  - recursão direta: a função A chama a própria função A
  - recursão indireta: a função A chama uma função B que, por sua vez, chama A

Condição de parada

- Nenhum programa nem função pode ser exclusivamente definido por si
  - Um programa seria um loop infinito
  - Uma função teria definição circular
- Condição de parada
  - Permite que o procedimento pare de se executar
  - *F(x)* > 0 onde *x* é decrescente
- Objetivo
  - Estudar recursividade como ferramenta *prática*!

Recursividade

- Para cada chamada de uma função, recursiva ou não, os parâmetros e as variáveis locais são empilhados na pilha de execução.

Execução

- Internamente, quando qualquer chamada de função é feita dentro de um programa, é criado um **Registro de Ativação** na **Pilha de Execução** do programa
- O registro de ativação armazena os parâmetros e variáveis locais da função bem como o "ponto de retorno" no programa ou subprograma que chamou essa função.
- Ao final da execução dessa função, o registro é desempilhado e a execução volta ao subprograma que chamou a função

Cálculo de fatorial com recursividade

```
#include <stdio.h>
    int fatorial( int n )
    {
      int i,p;
      p = 1;
      for( i=2; i<=n; i++ )
          p = p * i;
      return p;
    }
```

Cálculo de fatorial com recursividade

```
#include <stdio.h>

int main(){
     unsigned int numero;
     printf("\nEntre com um numero positivo.");
     scanf("%u", &numero);
     printf("O fatorial de %u vale %u.", numero,
fatorial(numero));
     system(“pause”);
}

unsigned int fatorial (unsigned int num){
    unsigned int fat;
   if (num == 1)
      return (1);
    else
        fat = num * fatorial (num-1);
    return fat;
}
```

Cálculo de fatorial com recursividade

```
factorial(6)
    6 * factorial(5)
    6 * 5 * factorial(4)
    6 * 5 * 4 * factorial(3)
    6 * 5 * 4 * 3 * factorial(2)
    6 * 5 * 4 * 3 * 2 * factorial(1)
    6 * 5 * 4 * 3 * 2 * 1 * factorial(0)
    6 * 5 * 4 * 3 * 2 * 1 * 1
    6 * 5 * 4 * 3 * 2 * 1
    6 * 5 * 4 * 3 * 2
    6 * 5 * 4 * 6
    6 * 5 * 24
    6 * 120
    720
```

Quando vale a pena usar recursividade

- Recursividade vale a pena para Algoritmos complexos, cuja a implementação iterativa é complexa e normalmente requer o uso explícito de uma pilha
  - Dividir para Conquistar (Ex. Quicksort)
  - Caminhamento em Árvores (pesquisa, backtracking)
- Vantagens da recursão
  - Redução do tamanho do código fonte
  - Maior clareza do algoritmo para problemas de definição naturalmente recursiva
- Desvantagens da recursão
  - Baixo desempenho na execução devido ao tempo para gerenciamento das chamadas
  - Dificuldade de depuração dos subprogramas recursivos, principalmente se a recursão for muito profunda

Função de Potência Recursiva

```
int pot(int base, int exp)
{
  if (!exp)
    return 1;

  /* else */
  return (base*pot(base, exp-1));
}
```

Exercícios

- Implemente uma função recursiva para computar o valor de 2^n

Respostas

```
Pot(int n) {
    if (n==0)
        return 1;
    else
        return 2 * Pot(n-1);
}
```