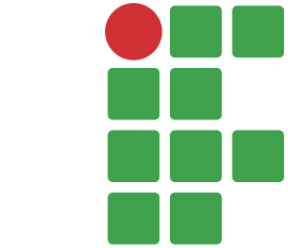
RECURSIVIDADE



INSTITUTO FEDERAL
Paraíba

Professor Msc Paulo de Tarso F. Júnior

paulodt@gmail.com

Roteiro

- Recursividade
  - Conceitos
  - Condição de Parada
  - Execução
  - Exemplos

Conceito

- 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

- 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

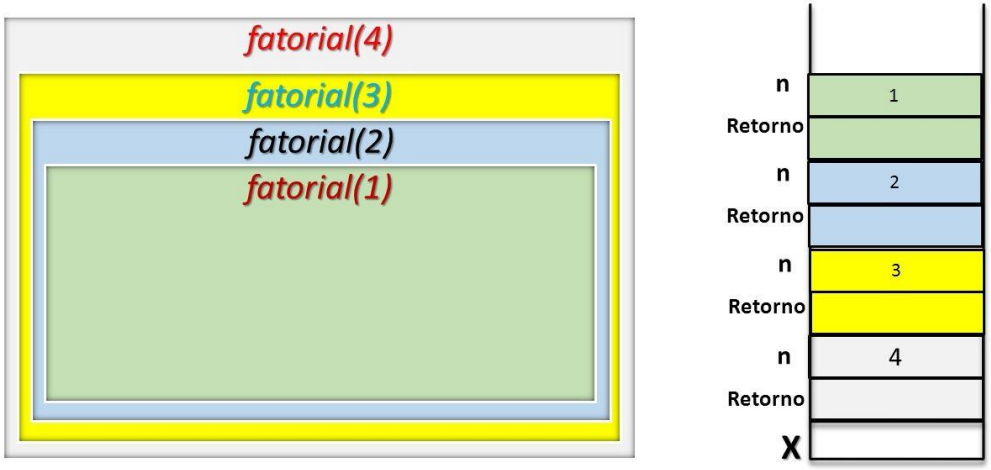
Condição de Parada

- 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.

Recursão e a 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.

Exemplo

```
int fat1(int n) {
 int r;
 if (n<=0)
  r = 1;
 else
  r = n*fat1(n-1);
 return r;
}

void main() {
 int f;
 f = fatX(4);
 printf("%d",f);
}
```

```
int fat2(int n) {
 if (n<=0)
  return 1;
 else
  return n * fat2(n-1);
}
```

Complexidade

- A complexidade de tempo do fatorial recursivo é O(n).
- Mas a complexidade de espaço também é O(n), devido a pilha de execução
- Enquanto no fatorial não recursivo a complexidade de espaço é O(1)

```
Fat (int n) {
    int f;
    f = 1;
    while(n > 0){
      f = f * n;
      n = n - 1;
    }
    return f;
}
```

Complexidade

- Portanto, a recursividade nem sempre é a melhor solução, mesmo quando a definição matemática do problema é feita em termos recursivos

Fibonacci

- Outro exemplo: Série de Fibonacci:
  - Fn = Fn-1 + Fn-2 n > 2,
  - F0 = 0 F1 = 1
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

```
int Fib(int n) {
    if (n<=0)
        return 0;
    else if ( n == 1)
        return 1;
    else
        return Fib(n-1) + Fib(n-2);
}
```

Fibonacci não recursivo

```
int FibIter(int n) {
    int i, k, F;

    i = 1; F = 0;
    for (k = 1; k <= n; k++) {
        F += i;
        i = F - i;
    }
    return F;
}
```

Complexidade

- 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. Mergesort)
  - Caminhamento em Árvores (pesquisa, backtracking)

Dividir para conquistar

- Duas chamadas recursivas
  - Cada uma resolvendo a metade do problema
- Muito usado na prática
  - Solução eficiente de problemas
  - Decomposição
- Não se reduz trivialmente como fatorial
  - Duas chamadas recursivas
- Não produz recomputação excessiva como Fibonacci
  - Porções diferentes do problema

Exercícios

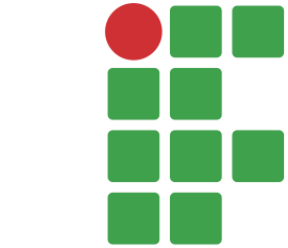
1. Crie uma função recursiva que calcula a potência de um número:
   - Como escrever a função para o termo n em função do termo anterior?
   - Qual a condição de parada?
2. Implemente uma função recursiva para computar o valor de 2^n
3. O que faz a função abaixo?

```
void f(int a, int b) { // considere a > b
    if (b == 0)
        return a;
    else
        return f(b, a % b);
}
```

Referências

- Material baseado em aula do professor David Menotti - DInf – UFPR

RECURSIVIDADE



INSTITUTO FEDERAL
Paraíba

Professor Msc Paulo de Tarso F. Júnior

paulodt@gmail.com