

Recursividade

Funções Recursivas

Prof. Leandro Colevati

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

Definição

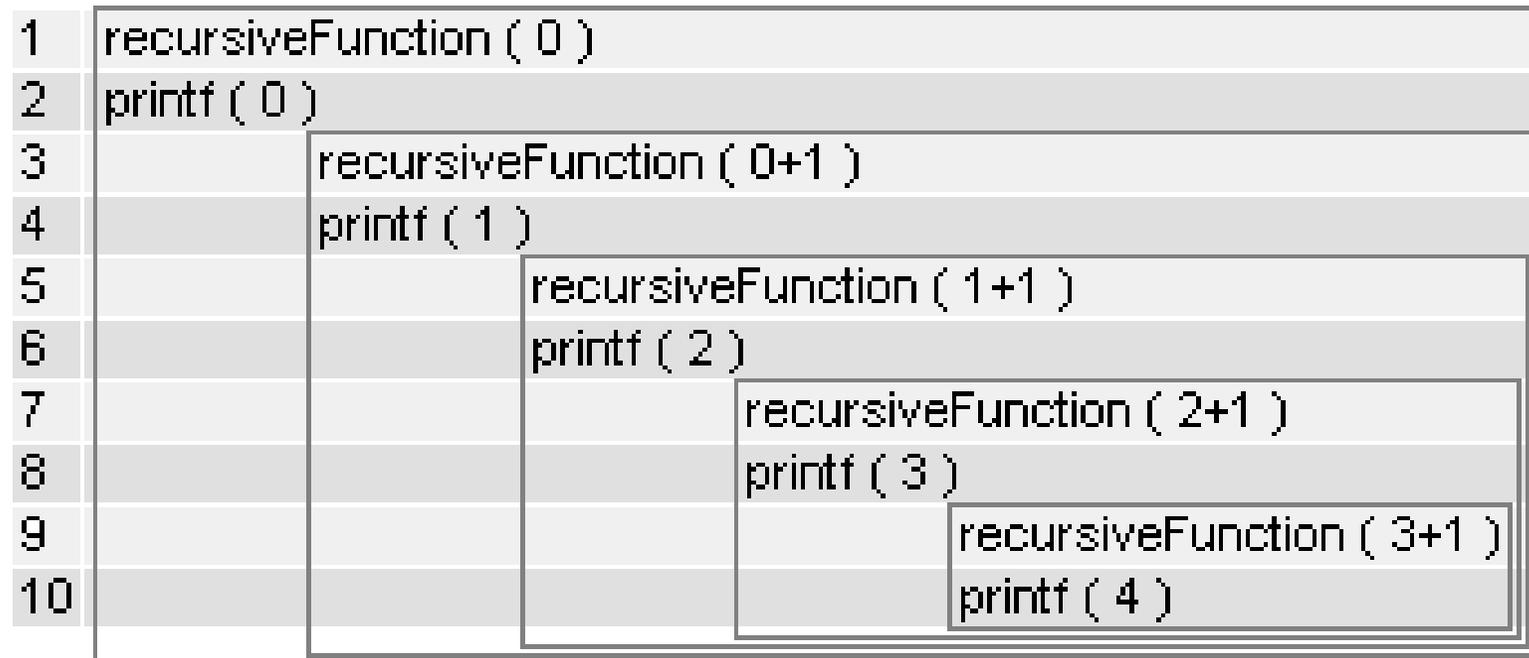
- Definição: dentro do corpo de uma função, chamar novamente a própria função
 - Recursão direta: a função A chama a própria função A
 - Recursão indireta: a função A chama uma função B que, por sua vez, chama A

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 de execução empilhada



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

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
- Já no fatorial não recursivo a complexidade de espaço é $O(1)$
- Portanto, a recursividade nem sempre é a melhor solução, mesmo quando a definição matemática do problema é feita em termos recursivos

Quando usar

- 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)

Exemplo 1

- Crie uma função recursiva que calcula a potência de um número inteiro:
 - Qual a condição de parada?
 - Como escrever a função para o termo n em função do termo anterior?

Exemplo 1

Algoritmo "Potencia Recursiva"

Var

base, expoente, resultado: inteiro

Funcao potencia(base, expoente : inteiro) : inteiro

Inicio

se (expoente = 0) entao

retorne 1

senao

retorne base * potencia(base, expoente - 1)

fimse

Fimfuncao

Inicio

base <- 2

expoente <- 8

resultado <- potencia(base, expoente)

escreval(resultado)

Fimalgoritmo

Exemplo 1

```
package controller;

public class PotenciaController {

    public PotenciaController() {
        super();
    }

    public int potencia(int base, int expoente) {
        //Condição de parada
        if (expoente == 0) {
            return 1;
        } else {
            expoente = expoente - 1;
            return base * potencia(base, expoente);
        }
    }

    public int pot(int base, int expoente) {
        int cont = 0;
        int res = 1;
        while (cont < expoente) {
            res = res * base;
            cont++;
        }
        return res;
    }
}
```

```
package view;

import controller.PotenciaController;

public class Principal {

    public static void main(String[] args) {
        PotenciaController pc = new PotenciaController();

        int base = 2;
        int expoente = 8;

        int pot = pc.pot(base, expoente);
        System.out.println("Com Laço ==> "+pot);

        int potencia = pc.potencia(base, expoente);
        System.out.println("Com recursividade ==> "+potencia);
    }
}
```

Exemplo 2

- Crie uma função recursiva que exiba o resultado da divisão de um número inteiro baseado em subtrações:
 - Qual a condição de parada?
 - Como escrever a função para o termo n em função do termo anterior?

Exemplo 2

Algoritmo "Divisão Recursiva"

Var

dividendo, divisor, resultado : inteiro

Funcao divisao(dividendo, divisor : inteiro) : inteiro

Inicio

se (dividendo < divisor) entao

retorne 0

senao

dividendo <- dividendo - divisor

retorne 1 + divisao(dividendo, divisor)

fimse

Fimfuncao

Inicio

dividendo <- 19

divisor <- 4

resultado <- divisao(dividendo, divisor)

escreval(resultado)

Fimalgoritmo

Exemplo 2

```
package controller;

public class DivisaoController {

    public DivisaoController() {
        super();
    }

    public int div(int dividendo, int divisor) {
        //Condição de parada
        if (dividendo < divisor) {
            return 0;
        } else {
            dividendo = dividendo - divisor;
            return 1 + div(dividendo, divisor);
        }
    }
}
```

```
package view;

import controller.DivisaoController;

public class Principal {

    public static void main(String[] args) {
        DivisaoController dc = new DivisaoController();

        int dividendo = 0;
        int divisor = 0;
        int div = dc.div(dividendo, divisor);
        System.out.println(div);
    }
}
```

Exemplo 3

- Crie uma função recursiva que exiba o resultado da soma das posições de um vetor de inteiros:
 - Qual a condição de parada?
 - Como escrever a função para o termo n em função do termo anterior?

Exemplo 3

- Visualg não aceita vetores como parâmetros

```
package controller;

public class VetorController {

    public VetorController() {
        super();
    }

    public int somaVetor(int[] vetor, int tamanho) {
        //Condição de parada ==> Quando o vetor não tiver mais posições
        if (tamanho == 0 ) {
            return 0;
        } else {
            int ultimaPosicao = tamanho - 1;
            int valor = vetor[ultimaPosicao];
            tamanho = tamanho - 1;
            return valor + somaVetor(vetor, tamanho);
            //Retorno é dado pelo valor obtido da última
            //posição do vetor, somado com a chamada da função
            //com um vetor de tamanho reduzido em 1 posição
        }
    }
}
```

```
package view;

import controller.VetorController;

public class Principal {

    public static void main(String[] args) {

        int[] vetor = {4,8,9,5,3,7,10,7,9,1,2};
        int tamanho = vetor.length;

        VetorController vc = new VetorController();
        int somaVetor = vc.somaVetor(vetor, tamanho);
        System.out.println(somaVetor);
    }
}
```