UNIRIO

Pilha de chamadas e Recursão

Aula 1

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

21 de agosto de 2018

Funções

Sequências de instruções agrupadas em unidades

- Também conhecidas como: procedimentos, rotinas, subrotinas, métodos, ...
- Trecho de código que pode ser invocado em outros trechos como uma etapa de uma computação maior
- Toda função possui: nome, parâmetros, tipo de retorno, corpo

```
public static int contaPares( int [] array ) {
    int numPares = 0;
    for( int numero : array )
        if( numero % 2 == 0 )
            numPares++;
    return numPares;
}
```

Funções

- A execução de uma função exige uma chamada
  - Função que chama deve fornecer argumentos para os parâmetros
- Corpo da função somente pode utilizar variáveis globais, variáveis locais e parâmetros
  - Não pode usar variáveis da função que chama: a função chamada não sabe quem foi que a chamou!
- Função que chama é interrompida na chamada
  - Transferência de controle
- Ao final de sua execução, função chamada deixa "para trás" uma variável de retorno
  - Execução é uma "caixa preta" para a função que chama: entram argumentos, sai retorno

Funções

Exemplo:

```
public static int contaPares( int [] array ) {
    int numPares = 0;
    for( int numero : array )
        if( numero % 2 == 0 )
            numPares++;
    return numPares;
}
public static void main( String [] args ) {
    int [] resultadosRoleta = {4, 6, 1, 3, 2, 7, 6, 2};
    int totalPontos = contaPares( resultadosRoleta );
}
```

Funções

Exemplo:

```
public static int contaPares( int [] array ) {
    int numPares = 0;
    for( int numero : array )
        if( numero % 2 == 0 )
            numPares++;
    return numPares;
}
public static void main( String [] args ) {
    int [] resultadosRoleta = {4, 6, 1, 3, 2, 7, 6, 2};
    int totalPontos = contaPares( resultadosRoleta );
}
```

Funções

Exemplo:

```
public static int contaPares( int [] array ) {
    int numPares = 0;
    for( int numero : array )
        if( numero % 2 == 0 )
            numPares++;
    return numPares;
}
public static void main( String [] args ) {
    int [] resultadosRoleta = {4, 6, 1, 3, 2, 7, 6, 2};
    int totalPontos = 5;
}
```

Para que utlizar funções?

- Reutilização de código
  - Uma rotina somente precisa ser implementada uma vez, mesmo que seja chamada em múltiplos pontos do código
- Melhor manutenção de código
  - Facilita o debug, evita erros do tipo copia-e-cola
- Melhor modularização
  - Isola tarefas diferentes em trechos diferentes de código, tornando-o mais compreensível
  - Permite esconder detalhes de implementação

Pilha de chamadas

Qualquer função pode chamar outra função:

Pilha de chamadas: sequência de funções, onde a do "topo" está sendo executada, e as outras esperam a de "cima" terminarem

Exemplo:

```
public static int potencia(int base, int expoente) {
}
```

Pilha de chamadas

Qualquer função pode chamar outra função:

Pilha de chamadas: sequência de funções, onde a do "topo" está sendo executada, e as outras esperam a de "cima" terminarem

Exemplo:

```
public static int potencia(int base, int expoente) {
    int totalMult = 1;
    for( int fator = 0; fator < expoente; fator++ ) {
        int totalFator = 0;
        for( int parcela = 0; parcela < base; parcela++ )
            totalFator += totalMult;
        totalMult += totalFator;
    }
    return totalMult;
}
```

Pilha de chamadas

Qualquer função pode chamar outra função:

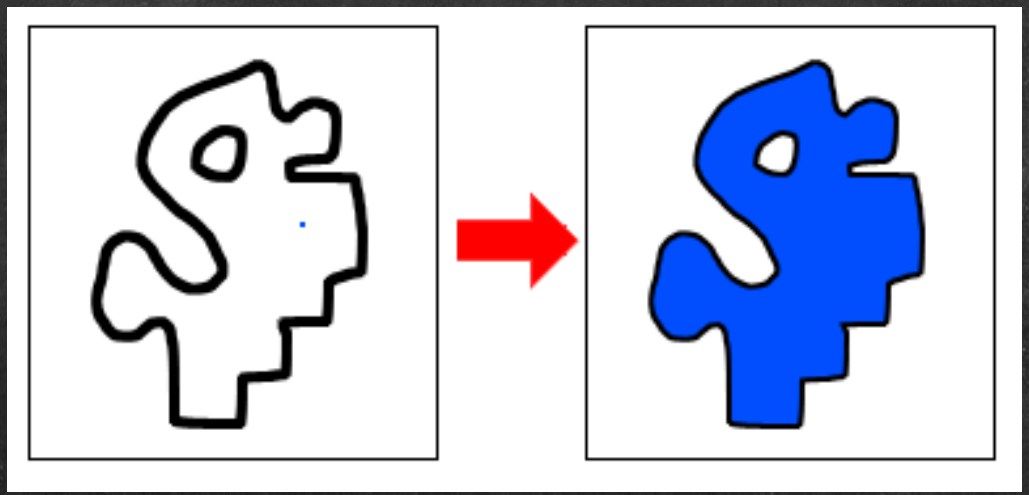
Pilha de chamadas: sequência de funções, onde a do "topo" está sendo executada, e as outras esperam a de "cima" terminarem

Exemplo:

```
public static int potencia(int base, int expoente) {
    int totalMult = 1;
    for( int fator = 0; fator < expoente; fator++ ) {
        totalMult = multiplica( base, totalMult );
    }
    return totalMult;
}
public static int multiplica( int fator1, int fator2 ) {
    int totalFator = 0;
    for( int parcela = 0; parcela < fator1; parcela++ ) {
        totalFator += fator2;
    }
    return totalFator;
}
```

Recursão

Desafio: criar uma função que execute o comando "preencher" ("flood fill") em uma imagem (como no Paint)



(retirado de Invent with Python)

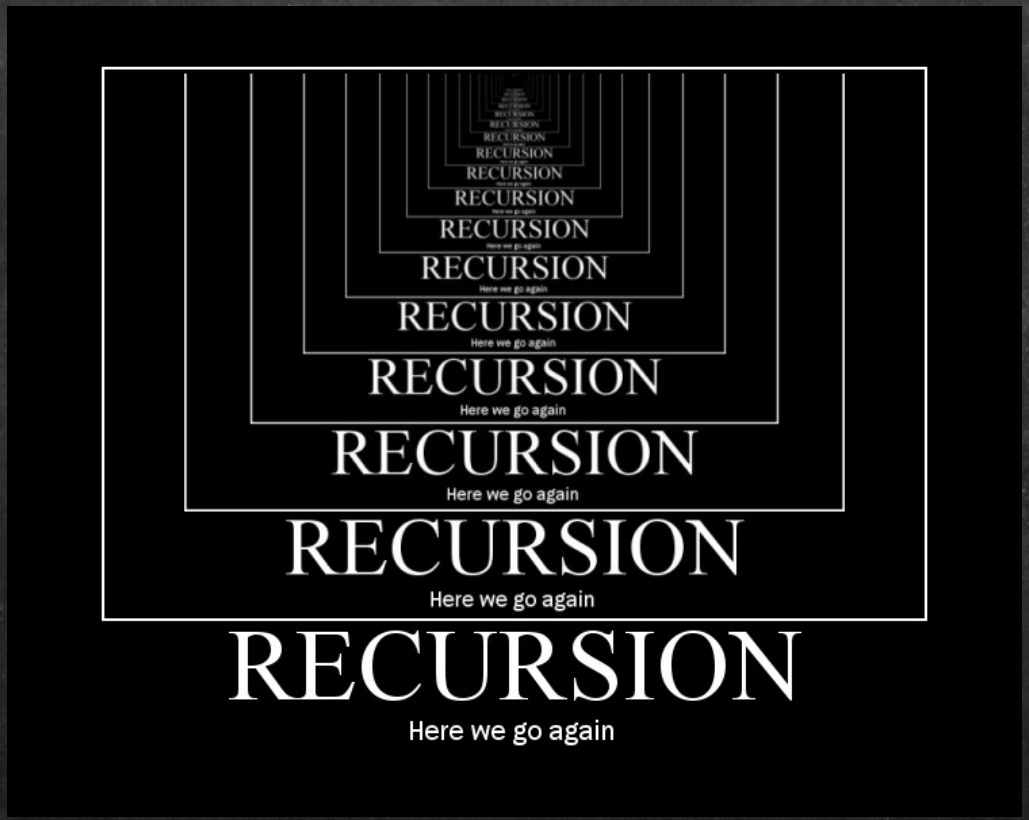
```
public static void preencher (int [][] imagem, int x, int y, int cor)
```

Recursão

Para entender recursão, você deve primeiro entender recursão.

Recursão

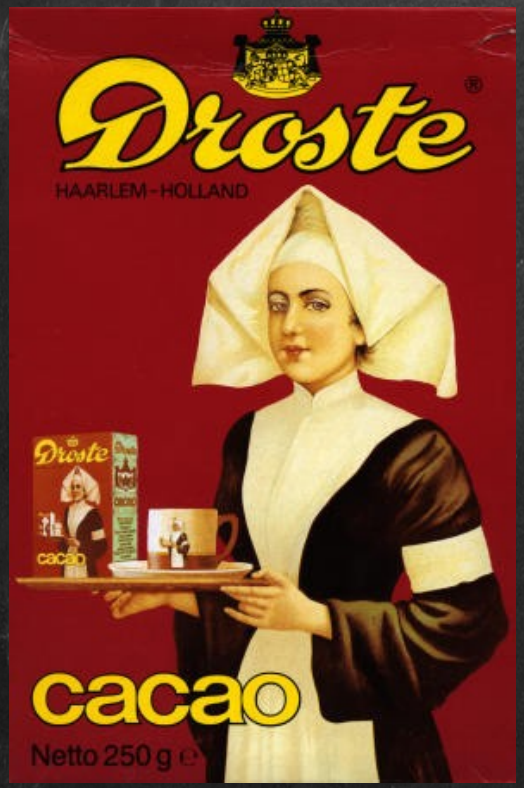
Para entender recursão, você deve primeiro entender recursão.

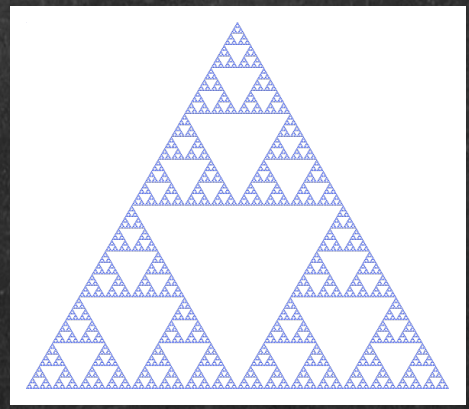


(retirado de Perpetual Enigma)

Recursão

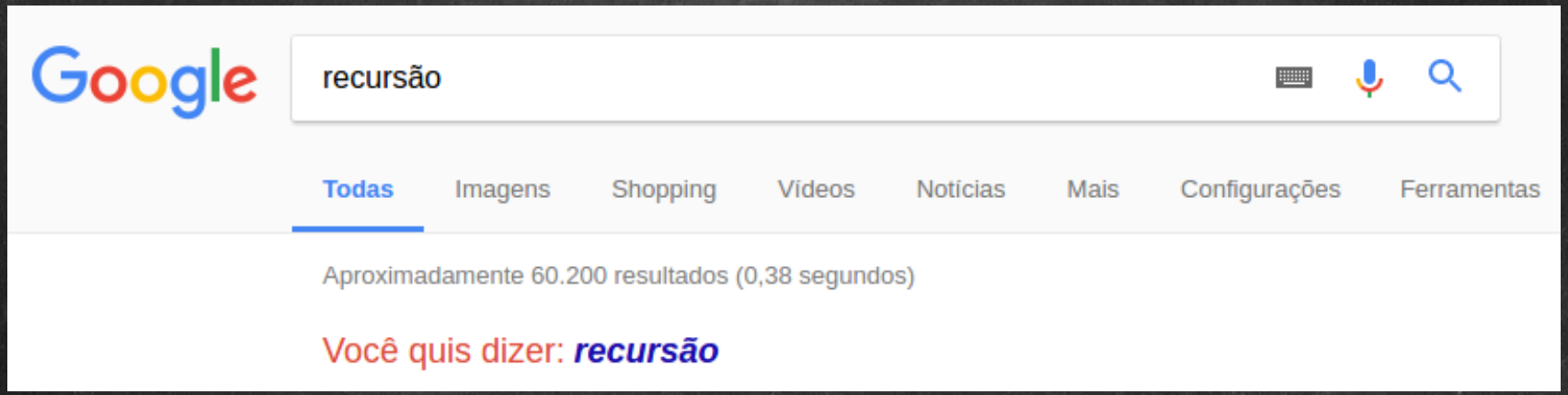
Recursão ocorre quando uma definição ou um procedimento invoca a si própria.
Exemplos:





(retirados de Wikipedia)

Recursão



- Recursão aparece em várias áreas de conhecimento
- Muito utilizada em matemática e computação
  - Formulação natural em diversos contextos
  - Exemplo: algoritmo de PageRank do Google

Recursão e programação

Forma mais comum: função que chama a si mesma

- Como a iteração, mas sem um "contador" explícito

Exemplo canônico: função fatorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$$

Recursão e programação

Forma mais comum: função que chama a si mesma

- Como a iteração, mas sem um "contador" explícito

Exemplo canônico: função fatorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n \cdot (n-1)!$$

Recursão e programação

Forma mais comum: função que chama a si mesma

- Como a iteração, mas sem um "contador" explícito

Exemplo canônico: função fatorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n \cdot (n-1)!$$

```
public static int fatorial (int n) {
        return n * fatorial(n-1);
}
```

Recursão e programação

Forma mais comum: função que chama a si mesma

- Como a iteração, mas sem um "contador" explícito

Exemplo canônico: função fatorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n \cdot (n-1)!$$

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

Recursão e programação

Cuidados fundamentais:

- Recursões precisam terminar!
  - Toda recursão precisa ter um caso base (ex: 0!)
  - Sequência de recursões precisa "caminhar" para o caso base
- Recursões podem ser longas demais
  - Pode exceder o tamanho da pilha de chamadas
- Em alguns casos, algoritmos recursivos podem ser reescritos de maneira iterativa

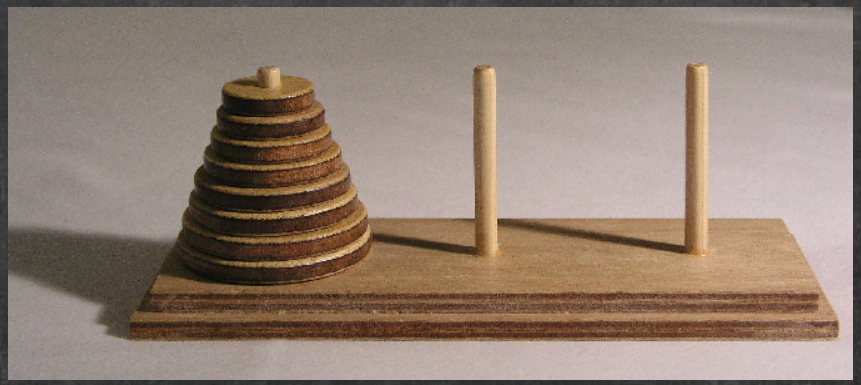
Recursão e programação

Cuidados fundamentais:

- Recursões precisam terminar!
  - Toda recursão precisa ter um caso base (ex: 0!)
  - Sequência de recursões precisa "caminhar" para o caso base
- Recursões podem ser longas demais
  - Pode exceder o tamanho da pilha de chamadas
- Em alguns casos, algoritmos recursivos podem ser reescritos de maneira iterativa

APRECIE COM MODERAÇÃO!

Torre de Hanói



Objetivo: mover todos os pinos de um pino para outro
Regras:

1. Todo movimento consiste em tirar um único disco de um pino e colocá-lo em outro
2. Discos maiores não podem ficar em cima de discos menores

Preencher

```
public static void preencher (int [][] imagem, int x, int y, int cor) {
}
```

Preencher

```
public static void preencher (int [][] imagem, int x, int y, int cor) {
    if( imagem[x][y] != cor ) {
        imagem[x][y] = cor;
    }
}
```

Preencher

```
public static void preencher (int [][] imagem, int x, int y, int cor) {
    int corOriginal = imagem[x][y];
    if( imagem[x][y] != cor ) {
        imagem[x][y] = cor;
        if( x > 0 && imagem[x-1][y] == corOriginal )
            preencher( imagem, x-1, y, cor );
    }
}
```

Preencher

```
public static void preencher (int [][] imagem, int x, int y, int cor) {
    int corOriginal = imagem[x][y];
    if( imagem[x][y] != cor ) {
        imagem[x][y] = cor;
        if( x > 0 && imagem[x-1][y] == corOriginal )
            preencher( imagem, x-1, y, cor );
        if( y < 0 && imagem[x][y-1] == corOriginal )
            preencher( imagem, x, y-1, cor );
        if( x < imagem.length - 1 && imagem[x+1][y] == corOriginal )
            preencher( imagem, x+1, y, cor );
        if( y < imagem[x].length - 1 && imagem[x][y+1] == corOriginal )
            preencher( imagem, x, y+1, cor );
    }
}
```

Exercícios

Crie funções para realizar recursivamente as seguintes tarefas:

1. Calcular o n-ésimo número de Fibonacci
   - A sequência de Fibonacci começa com 0 e 1, e cada número é igual a soma dos dois anteriores

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

2. Calcular o máximo divisor comum entre dois inteiros positivos
   - Use o método de Euclides: se $a > b$ e b não divide a, então $MDC(a, b) = MDC(b, a \mod b)$

Tente também resolver os problemas 1 e 2 de maneira iterativa!

Exercícios

Crie funções para realizar recursivamente as seguintes tarefas:

3. Verificar se uma palavra é um palíndromo
   - Dica: para pegar partes de uma String, utilize a função `substring()`
4. Gerar a forma binária (string de 0's e 1's) de um número inteiro
5. Buscar um número em um array de inteiros ordenados
   - (este algoritmo é conhecido: chama-se Binary Search; tente resolver sem consultá-lo!)
6. Resolver o problema da Torre de Hanói

Tente também resolver os problemas 4 e 5 de maneira iterativa!