Programação Estruturada

Aula 09 - Recursão

Apresentação

Na aula passada, você aprendeu a programar de forma modularizada, criando procedimentos e funções e viu que com esses recursos é possível dividir grandes problemas em problemas menores, mais fáceis de resolver. Essa divisão é uma das alternativas mais utilizadas para desenvolver grandes programas.

Nesta aula, você irá estudar sobre o uso de funções e procedimentos de forma recursiva. A recursão, como veremos, é um importante aliado para simplificar a programação de determinadas rotinas.

Faça uma boa leitura!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

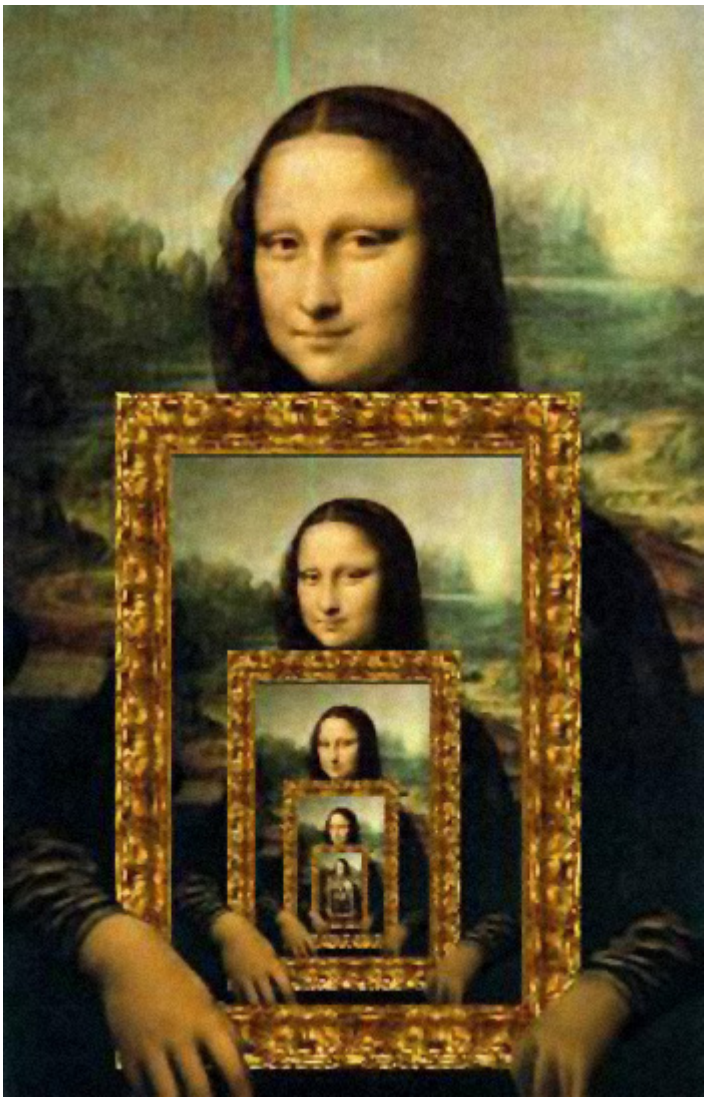
- Descrever o funcionamento de funções e procedimentos recursivos
- Identificar quando devemos implementar funções e procedimentos de forma recursiva.

1. Introdução

O termo recursão é bastante utilizado entre os programadores de computador, principalmente entre os mais experientes. Esse fato deve-se a dois motivos. O primeiro deles é que a recursão é uma técnica poderosa que facilita a programação de funções e procedimentos em determinadas situações específicas. Esse assunto é mais comentado entre os programadores mais experientes porque se leva, geralmente, um tempo maior para conseguir assimilar a ideia por traz da recursão e para usá-la corretamente na prática. Por isso, tenha bastante atenção ao assunto que veremos aqui, tudo bem?

Comecemos entendendo melhor a definição do conceito de recursão. A recursividade é um termo bastante genérico, usado não só na computação. Ela remete ao processo de repetição de algo (um objeto, uma declaração etc.) que já fora apresentado anteriormente. Veja isso de forma simples, através de figuras do dia a dia. Você consegue perceber alguma recursividade na figura mostrada a seguir?

Figura 01 - Quadro que mostra o próprio quadro



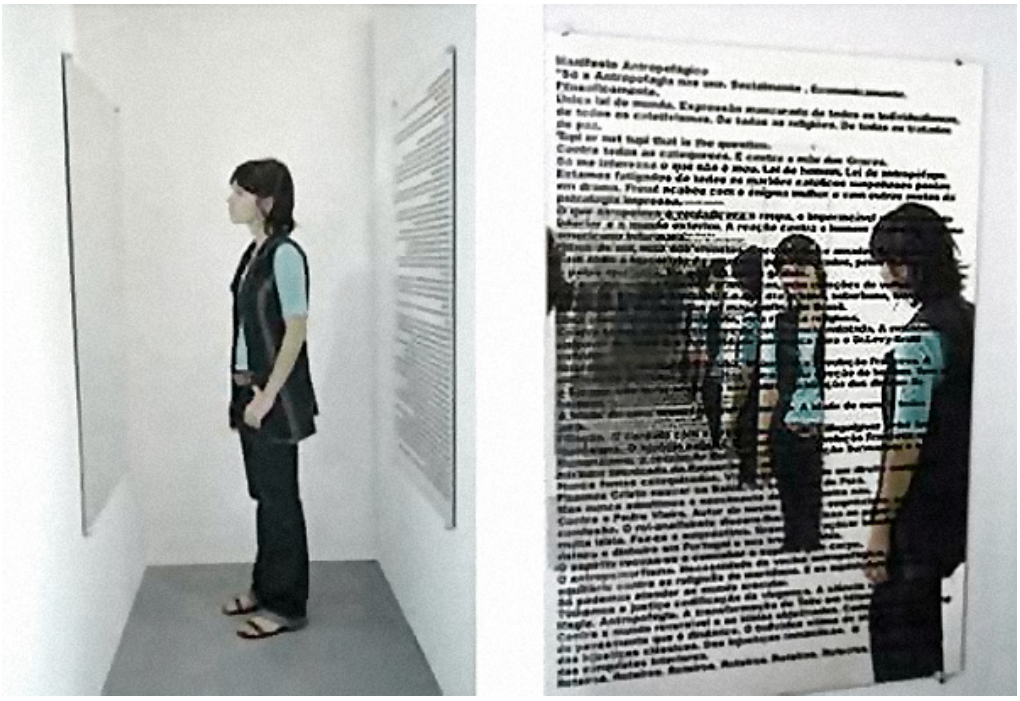
Fonte: http://ldev.wordpress.com/2009/03/16/recursividade/. Acesso em: 30 set. 2011.

Ficou fácil, não foi? A figura representa uma pintura de um quadro que aparece novamente dentro da própria figura. Se ele aparece dentro da própria figura, isso configura a recursão. Note que essa recursão é infinita! Se tivermos uma lupa, dentro de cada quadro da Figura 1 iremos encontrar um quadro menor dentro dele, o qual possui outro quadro dentro dele menor ainda, e por aí vai. Claro que se esse processo é feito por um ser humano, chegará um momento que ele não terá precisão para pintar quadros de tamanho ínfimo.

Um exemplo mais comum que você já deve ter percebido é quando estamos em um local com um espelho na frente e outro atrás da gente. Sua imagem, que é refletida pelo espelho da frente, é refletida pelo espelho de trás. Este segundo

reflexo aparece no espelho da frente como um clone seu, que é refletido novamente para o espelho de trás, que por sua vez projeta mais um clone seu no espelho da frente e por aí vai.

Figura 02 - Recursão criada por conjunto de espelhos



Fonte:
http://www.muvi.advant.com.br/artistas/r/rosana_ricalde/exercicio_da_possibilidade.htm.
Acesso em: 10 out. 2011.

2. Recursão na Programação de Computadores

Durante esta aula, veremos que a recursão pode ser usada na implementação de diversas funções e procedimentos. Vamos ver como aplicamos esse conceito? Observe o seguinte programa:

```
1 public class ProgramaRecursaoInfinita {
2    public static void main(String[] args) {
3       System.out.println(funcaoX());
4    }
5
6    public static int funcaoX() {
7       return funcaoX();
8    }
9 }
```

No programa mostrado, temos a definição de uma função de nome funcaoX(). No corpo dessa funcaoX() temos uma chamada para a própria funcaoX() o que torna ela uma função recursiva. Porém, se percebe que essa função nunca irá retornar nada, pois sempre é chamada novamente antes do retorno. Temos então um caso de uma recursão infinita, como aquela do espelho. Entenda o que está acontecendo: quando a rotina main chama a funcaoX(), o corpo dessa função chama novamente a própria funcaoX(). Essa segunda execução da funcaoX() irá chamar uma terceira execução da funcaoX() e assim por diante.

Vídeo 02 - Funções e Procedimentos Recursivos

Rodar um programa como o *ProgramaRecursaoInfinita* irá implicar no encerramento do seu programa por falta de memória. Isso porque toda vez que uma função ou procedimento é chamado, um espaço na memória é alocado para armazenar informações, como os valores das variáveis locais etc. Em uma recursão infinita você chama a própria função quantas vezes? Infinitas vezes, certo? Como a memória é finita, chegará um momento que não haverá mais memória para ser alocada e o programa será interrompido pelo sistema operacional do computador.

Muito bem, essa é a forma errada de se usar recursão em programação. O problema do *ProgramaRecursaoInfinita* é que a recursão implementada nele nunca parar e toda função ou procedimento recursivo uma hora tem que parar. Para que isso aconteça, você geralmente vai precisar fazer um teste dentro da rotina recursiva.

Mas vejamos um exemplo prático de função que é melhor implementada através de recursão. O fatorial de um número natural n é uma função matemática representada pela notação n! e é calculado pelo produto de todos os números inteiros positivos menores ou iguais a n. Na notação matemática, temos o fatorial definido por:

$$n! = \prod_{k=1}^{n} k \; \forall n \in \mathbb{N}$$

(Equação 1)

Veja como esse valor é calculado no caso do fatorial de 5:

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

(Equação 2)

Um caso especial é o do número 0 (zero):

$$0! = 1$$

(Equação 3)

O fatorial não é calculado para números negativos, os quais não fazem parte dos números naturais. Sabendo disso, como você poderia implementar um programa para calcular o fatorial de um número? Veja uma possível implementação a seguir.

```java
import java.util.Scanner;

public class ProgramaFatorialIterativo {

   public static void main(String[] args) {
      Scanner leitor = new Scanner(System.in);
      System.out.println("Digite um número:");
      int numero = leitor.nextInt();
      int fat = fatorial(numero);
      if (fat > 0) {
         System.out.println("O fatorial desse número é " + fat);
      } else {
         System.out.println("Não existe fatorial para números negativos!");
      }
   }

   public static int fatorial(int numero) {
      int fat;
      if (numero < 0) {
         fat = -1;
      } else if (numero == 0) {
         fat = 1;
      } else {
         fat = 1;
         for (int i = 2; i <= numero ; i++) {
            fat = fat * i;
         }
      }
      return fat;
   }

}
```

Na rotina *main* do *ProgramaFatorialIterativo*, é lido um número (variável numero) e é declarada e utilizada a função **fatorial()** para se calcular o fatorial desse número digitado. A função *fatorial()* é organizada através de um comando de seleção *if*. Isso porque existem três casos que precisam ser verificados. O primeiro é se o número passado como parâmetro é negativo (numero < 0). Se for o caso, a função está retornando o valor -1 para indicar o erro. Na disciplina de Programação Orientada a Objetos, você verá como tratar melhor esses casos de erro através do uso de exceções.

Vídeo 03 - Loop Infinito

O segundo caso verificado é o do número digitado ser igual a 0 (numero == 0). Nessa situação, a função não precisa calcular nada, apenas retornar o valor 1, que por definição é o valor fatorial de zero. Já o último caso é aquele em que se calcula realmente o fatorial de acordo com a equação matemática mostrada para o fatorial. Veja que utilizamos o comando de iteração *for* para realizar a multiplicação do cálculo do fatorial:

```
1 fat = 1;
2 for (int i = 2; i <= numero ; i++) {
3    fat = fat * i;
4 }
```

Dado que o valor da variável *fat* é igual a 1 e que a variável *i* que indexa o laço é inicializada com o valor 2, está se fazendo um laço que, enquanto *i* não for maior que o número digitado, se atualiza o valor de *fat* multiplicando seu valor por *i*. Por exemplo, para o fatorial de 5, teremos o cálculo ilustrado no quadro a seguir, no qual temos a inicialização da variável *fat* com o valor 1, e depois uma sequência de laços multiplicando o valor da variável *fat* pelos números de 2 (valor inicial de *i*) a 5 (valor final de *i*).

```
fat = 1

Laço 1 (i = 2)
   fat = fat * i
      = 1 * 2

Laço 2 (i =3)
   fat = fat * i
      = (1 * 2) * 3

Laço 3 (i = 4)
   fat = fat * i
      = (1 * 2 * 3) * 4

Laço 4 (i = 5)
   fat = fat * i
      = (1 * 2 * 3 * 4) * 5
      = 120
```

Pois bem, essa forma de implementação do fatorial é válida, porém, não é a mais simples e natural possível. Para você entender isso, observe que o fatorial pode ser definido facilmente através das seguintes fórmulas:

$$\text{fatorial}(n) = \begin{cases} 1 & \text{se } n = 0 \\ x \times \text{fatorial}(n-1), & \text{se } n > 0 \end{cases}$$

(Equação 4)

Você notou alguma coisa especial nessa fórmula de cálculo para o fatorial de n? Ela possui uma definição recursiva! Observe que o fatorial de n é definido em termos do fatorial de n - 1. Baseado nessa nova definição, o fatorial pode ser calculado da forma mostrada a seguir. Note que para calcular o fatorial de 5, precisamos calcular o fatorial de (5 – 1), ou seja, o fatorial de 4, cujo valor depende do valor do fatorial de 3, e assim por diante, até chegarmos ao fatorial de 0, que já é definido como 1, como vimos na equação 3. Esse valor definido é o que faz a função fatorial ter um fim e é conhecido como **caso base**.

```
fatorial(5) = 5 * fatorial(5 − 1)
= 5 * fatorial(4)
= 5 * 4 * fatorial(4 - 1)
= 5 * 4 * fatorial(3)
= 5 * 4 * 3 * fatorial(3 - 1)
= 5 * 4 * 3 * fatorial(2)
= 5 * 4 * 3 * 2 * fatorial(2 − 1)
= 5 * 4 * 3 * 2 * fatorial(1)
= 5 * 4 * 3 * 2 * 1 * fatorial(1 − 1)
= 5 * 4 * 3 * 2 * 1 * fatorial(0)
= 5 * 4 * 3 * 2 * 1 * 1
= 120
```

Quando a solução do problema a ser resolvido pode ser apresentada de forma recursiva, isso indica que sua implementação via recursão é mais simples e mais apropriada. Veja a seguir a implementação do mesmo programa, porém agora implementando a função fatorial através de recursão.

```java
import java.util.Scanner;

public class ProgramaFatorialRecursivo {

    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite um número:");
        int numero = leitor.nextInt();
        int fat = fatorial(numero);
        if (fat > 0) {
            System.out.println("O fatorial desse número é " + fat);
        } else {
            System.out.println("Não existe fatorial para números negativos!");
        }
    }

    public static int fatorial(int numero) {
        int fat;
        if (numero < 0) {
            fat = -1;
        } else if (numero == 0) {
            fat = 1;
        } else {
            fat = fatorial(numero - 1) * numero;
        }
        return fat;
    }

}
```

Note que o que muda é apenas o código da função fatorial. Nesse código, existe a mesma estrutura de *if/else* que existe na versão iterativa da solução (uso do *for*). Entretanto, o corpo do último *else* – onde se calcula o valor do fatorial de números maiores que zero – foi simplificado. Isso não só reduz a quantidade de linhas de código do programa, mas também facilita a leitura e o entendimento do código por outras pessoas.

Vídeo 04 - Funções e Procedimentos Recursivos

Atividade 01

1. Implemente e execute o *ProgramaFatorialIterativo* e o *ProgramaFatorialRecursivo*. Relate se a execução dos dois programas se comporta da mesma forma para números negativos, zero e números positivos.

2. Descreva, com suas palavras, qual a forma mais interessante de implementação da função fatorial.

3. Implemente de forma recursiva a função de Fibonacci, que é dada pela seguinte fórmula:

$$F(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

3. Uso da Recursão ou da Iteração (laços)

Como dito anteriormente, a recursão é uma técnica poderosa para os programadores. Entretanto, ela não deve ser considerada um substituto da iteração. Isso porque nem sempre está claro se a recursão é a melhor opção de implementação para uma função ou procedimento. De fato, às vezes a implementação mais fácil de uma solução é a iterativa, usando-se *for* ou *while*.

Observe o código do programa mostrado a seguir.

```
import java.util.Scanner;

public class ProgramaRecursao1 {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite um número a ser dobrado várias vezes:");
        int numero = leitor.nextInt();
        System.out.println("Digite um número limite para a operação:");
        int limite = leitor.nextInt();
        System.out.println(dobrarNumeroAteLimite(numero, limite));
    }

    public static int dobrarNumeroAteLimite(int numero, int limite) {
        int dobro = 2 * numero;
        if (dobro >= limite) {
            return numero;
        } else {
            return dobrarNumeroAteLimite(dobro, limite);
        }
    }
}
```

Você consegue perceber o que esse código faz? Veja que são lidos dois números, um primeiro (variável numero), que representa um número a ser dobrado, ou seja, multiplicado por dois, e um segundo (variável limite), que indica o valor máximo. Veja a seguir a versão desse programa de forma iterativa, usando o *for*:

```
import java.util.Scanner;

public class ProgramaIterativoDobra {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite um número a ser dobrado várias vezes:");
        int numero = leitor.nextInt();
        System.out.println("Digite um número limite para a operação:");
        int limite = leitor.nextInt();
        System.out.println(dobrarNumeroAteLimite(numero, limite));
    }

    public static int dobrarNumeroAteLimite(int numero, int limite) {
        for (int aux = numero * 2; aux < limite; aux = numero * 2) {
            numero = aux;
        }
        return numero;
    }
}
```

No *ProgramaIterativoDobra*, existe uma variável auxiliar que fica recebendo em cada laço o dobro do valor da variável número. Enquanto esse valor não ultrapassar o valor limite, a variável número vai sendo atualizada, sendo seu valor retornado ao final da execução da função.

Vídeo 05 - Loop Controlado

Qual das duas versões do programa você acha mais interessante? Em termos de facilidade de entendimento do código, acreditamos que a versão recursiva é mais fácil de entender. Entretanto, a implementação de forma iterativa também é uma boa solução, cujo entendimento também é simples. Nesses casos, geralmente o que conta é a forma de pensar do programador. Quem tem mais facilidade com programação recursiva geralmente tende a usar mais recursão. Já quem tem mais facilidade com iteração, tende a programar mais com laços do que com recursão. O mais importante é você conhecer e saber utilizar as duas abordagens.

Vídeo 06 - Fibonnaci

Atividade 02

1. Implemente uma versão iterativa para a função de Fibonacci.
2. Descreva, com suas palavras, qual implementação foi mais fácil de se utilizar.

4. Resumo

Nesta aula, você aprendeu conceitos sobre recursão aplicada à programação, que no contexto desta disciplina é a implementação de funções e procedimentos usando a própria definição da função. Dessa forma, o corpo de uma função recursiva faz referências a ela própria, ou seja, durante sua execução, ela própria inicia uma ou mais vezes novas execuções dela mesma. Essa forma de implementação pode simplificar o desenvolvimento de alguns programas. Entretanto, apesar de poder na maioria das vezes usar recursão no lugar de iteração (laços *for, while* etc.), a recursão não pode ser vista como um substituto da iteração.

Para a próxima aula, você verá o desenvolvimento de um jogo utilizando-se recursão. Até lá!

5. Autoavaliação

1. Qual a ideia do uso da recursão na programação de funções e procedimentos? Como elas se diferenciam das outras funções?
2. Quando devemos optar pela recursão na implementação de uma função ou de um procedimento?

6. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.