Universidade Federal de Campina Grande – UFCG
Centro de Engenharia Elétrica e Informática – CEEI
Departamento de Sistemas e Computação – DSC
Disciplina: Programação 2

Exercício sobre recursividade

1. O que o código seguinte faz?

```
public class Misterio {
       public static void main(String[] args) {
        int x = //le inteiro do teclado;
        int y = //le inteiro do teclado;
        System.out.println(misterio(x, y));
    }
    private static int misterio(int x, int y){
        if (y==0){
               return 1;
        }
        return (x*misterio(x,y-1));
    }
}
```

2. Explique o que é passo de recursão e condição de parada (ou caso base). Na questão anterior, identifique estes termos.

3. Quando trabalhamos com recursão é preciso estar atento a algumas situações que podem gerar erros no código: esquecer o caso base (a), não convergir (b), muito consumo de memória (c), muita computação repetida (d) . Você consegue observar tais situações nos códigos abaixo?

public static double H(int N) {
  return H(N-1) + 1.0/N;
}
	public static double H(int N) {
	  if (N == 1) return 1.0;
	  return H(N) + 1.0/N;
	}

public static double H(int N) {
  if (N == 0) return 0.0;
  return H(N-1) + 1.0/N;
}
	public static long F(int n) {
	  if (n == 0) return 0;
	  if (n == 1) return 1;
	  return F(n-1) + F(n-2);
	}

4. A função a seguir calcula o MDC (máximo divisor comum) de dois inteiros positivos m e n. Escreva uma função recursiva equivalente.

```
public static int Euclides(int m, int n){
        int r;
        do{
                r = m%n;
                m = n;
                n = r;
        } while (r != 0);
        return m;
}
```

É interessante visualizar a "recursividade em ação".

a) Entenda o código recursivo a seguir que retorna o i-ésimo termo da série de Fibonacci.

https://en.wikipedia.org/wiki/Conga_line

Identifique a condição de parada/casos base e o passo de recursão.

```
public class Fibonacci {
       public long fibonacci(long number) {
              if ((number == 0) || (number == 1))
                     return number;
              return fibonacci(number - 1) + fibonacci(number - 2);
       }
}
```

b) Uma boa visualização do que acontece quando n = 3 está ilustrada na figura a seguir. Faça uma ilustração semelhante acompanhando a execução para o caso em que n = 5.

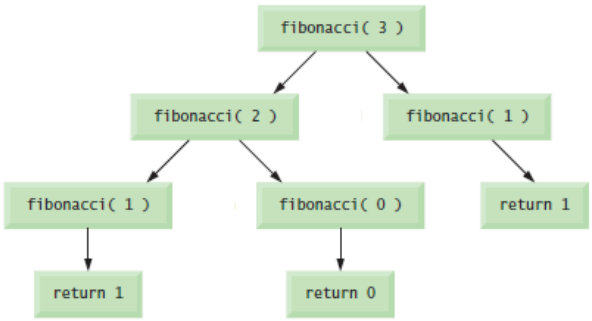

Figura 1 Passos da execução fibonacci(3)

c) Insira rótulos nos vértices da árvore gerada no item b) para indicar a ordem em que cada chamada ao método fibonacci ocorre. Por exemplo, a chamada fibonacci(5) terá rótulo 1.

d) Que tal mais esse desafio? Ilustre o passo a passo.
Qual a sequencia de números impresso na execução desafio(5):

```
public static void desafio(int n) {
        if (n <= 0) return;
        System.out.println(n);
        desafio(n-2);
        desafio(n-3);
        System.out.println(n);
}
```

6. Podemos escrever a recursão com OO introduzindo a ideia de objetos recursivos, ou seja, um objeto que tem como atributo um objeto do seu mesmo tipo. Nesse caso, a recursão ocorre quando um método chama ele mesmo, mas, sobre objetos diferentes.

a) Estude o código da dança CongaLine e veja como a recursividade OO foi implementada no método de inserção do Dancarino e no toString. O código está no seguinte github: https://github.com/liviamrs/p2lp2_20161. Depois de entender este código recursivo, tente pensar em dois métodos iterativos para varrer e inserir elementos na conga line.

b) O que é melhor? Usar uma solução iterativa ou recursiva? Uma é sempre melhor que a outra?

7. Pratique mais o conceito de recursão OO com os exercícios a seguir.

a) Considere a CongaLine do exemplo acima. Como seria um método recursivo para contar os dançarinos na lista?

b) Considere uma árvore binária referente a filiação. Deseja-se encontrar determinado parente. Escreva um método que retorna um valor booleano se uma pessoa faz parte da árvore ou não.

c) Considerando ainda a árvore de filiação, dado uma determinada pessoa, escreva um método que indique a qual geração pertence determinada pessoa. Se é a árvore de Joãozinho, ele é a geração 0, seus pais, a geração 1 e assim por diante.

8. Considere o código abaixo que representa uma caixa colorida. Note que é possível colocar uma caixa (apenas uma) em cima da outra. Diante disso, é possível criar uma pilha de caixas coloridas. Escreva

um método recursivo que sobe nessa pilha e retorna a posição da caixa da cor especificada como parâmetro. Se a caixa da cor especifica não estiver na pilha, o método deve retornar -1.

```
 public class CaixaColorida{

        private Cor cor;
        private CaixaColorida proxima;

        public CaixaColorida(Cor cor){
                this.cor = cor;
                this.proxima = null;
        }

        public CaixaColorida getProxima() {return proxima;}
        public Cor getCor() {return cor;}

        public void empilha(CaixaColorida outraCaixa){
                this.proxima = outraCaixa;
        }
}
public enum Cor {VERMELHO,PRETO,BRANCO,AZUL,VERDE;}
```