Exercícios: Estruturas de Repetição (2)

Importante

Utilize soluções iterativas e a estrutura de repetição mais apropriada à resolução de cada aspecto do problema (**`while`**, **`do-while`** e **`for`**). Participe das aulas com os exercícios progressivamente desenvolvidos, com dúvidas demarcadas e pontuais.

Exercício 1

Defina a classe **`InteiroMatematico`**. Objetos desta classe possuem, como principal atributo, um número inteiro ***n***, fornecido ao construtor. Esse atributo possui métodos tanto de acesso quanto de modificação. Sempre, entretanto, se assume **|*n*|** para internalizá-lo ao objeto.

A implementação dos demais métodos é descrita na sequência. Os parâmetros e o retorno precisam ser inferidos.

Não utilize classes externas para cumprir os exercícios. O foco deve ser o benefício didático de como resolver.

Exercício 2

`tabuada()`

Retorna uma cadeia de caracteres formatada com a tabuada de *n*.

Exercício 3

`divisores()`

Retorne, em ordem crescente, todos os divisores de *n*.

Exercício 4

`maiorDivisor()`

Retorne o maior divisor de *n*, exceto ele próprio. Se *n* = 0, ou *n* = 1, retorne 1.

Exercício 5

`menorDivisor()`

Retorne o menor divisor de *n*, exceto 1. Se *n* = 0, ou *n* = 1, retorne 1.

Exercício 6

`elevado()`

Retorne $n^{expoente}$. O expoente é outro número inteiro, passado por parâmetro, e deve ser positivo. Caso expoente <= 0, retorne 0. Em caráter de simplificação, desconsiderou-se que 0^0 é indeterminado. O cálculo precisa ser iterativo.

Exercício 7

`fatorial()`

Retorne *n!*, também calculado de maneira iterativa.

Exercício 8

`fibonacci()`

Retorne os *n* primeiros termos da série de Fibonacci. Saiba que os dois primeiros termos desta série são 1 e 1 e os demais são gerados a partir da soma dos anteriores: 1 1 2 3 5 8 13 21...

Exercício 9

`produtoPelaSoma()`

Retorne o produto de *m* por *n*, calculado por meio de somas sucessivas. O parâmetro *m* é outro número inteiro, considerado em módulo. Perceba a vantagem de tomar o menor dos dois números como multiplicador, pois calcular 2 x 10, nesse método, é menos custoso do que 10 x 2.

Exercício 10

`isTriangular()`

Retorne se *n* é um número triangular, ou seja, formado pelo produto de três inteiros consecutivos. Exemplos: 6 = 1 x 2 x 3; 24 = 2 x 3 x 4. Cuide com os passos desnecessários. Observe que o retorno deve ser booleano.

Exercício 11

`isSomaDosQuadrados()`

Repare a seguinte característica do número 3025: 30 + 25 = 55 e 55^2 = 3025.

Retorne se *n* possui essa mesma característica. Caso *n* seja menor do que 1.000 ou maior do que 9.999, retorne falso de imediato.

Exercício 12

`serieHarmonica()`

Calcule e retorne o valor de *H*, considerando que *n* representa o número de termos da série abaixo.

```
H = 1 + (1/2) + (1/3) + (1/4) + ... + (1/n)
```

Exercício 13

`mdc()`

O Máximo Divisor Comum (MDC) entre dois números *m* e *n* é o maior número inteiro encontrado, que seja fator dos outros dois. Por exemplo, *mdc(16, 8) = 8*. A definição abrange qualquer número de termos, por exemplo *mdc(m, n, o, p)*.

Calcule e retorne o *mdc(m, n)*, sendo *m* fornecido como parâmetro.

Exercício 14

`mdcEficiente()`

Retorne o *mdc(m, n)* calculado desta outra maneira:

```
mdc(36, 10)
a    b    resto
36   10   6
10   6    4
6    4    2
4    2    0

mdc = 2 = b
```

Observe que o resto da divisão de *a* por *b*, ao final, é igual a zero.

Exercício 15

`mmc()`

O Mínimo Múltiplo Comum (MMC) de dois inteiros, *m* e *n*, é o menor inteiro positivo múltiplo simultaneamente de *m* e de *n*. Se não existir tal inteiro positivo, por exemplo, se *m=0* ou *n=0*, então *mmc(m, n)* é zero por definição.

Calcule e retorne o *mmc(m, n)*, sendo *m* fornecido como parâmetro.

Exercício 16

`mmcEficiente()`

Retorne o *mmc(m, n)* calculado eficientemente:

```
mmc = m * n / mdc(m, n)
```

Exercício 17

`tribonacci()`

Retorna os *n* primeiros termos da série de Tribonacci. Um número Tribonacci assemelha-se a um número de Fibonacci, mas em vez de se começar com dois termos predefinidos, a sequência é iniciada com três termos já determinados, e cada termo posterior é a soma dos três termos precedentes: 1, 1, 2, 4, 7, 13, 24, 44, 81...

Exercício 18 (importantíssimo)

`isPrimo()`

Retorne se *n* é um número primo.

Exercício 19

`isRaizExata()`

Retorne se a raiz de *n* é exata. Para isso, subtraia de *n* os ímpares consecutivos a partir de 1, até que o resultado da subtração seja menor ou igual a zero. O número de vezes que se conseguir fazer a subtração é a raiz quadrada exata (resultado 0) ou aproximada de *n* (resultado negativo).

Observe:

```
Raiz de 16
16 – 1 = 15 – 3 = 12 – 5 = 7 – 7 = 0
     ^        ^        ^       ^ = 4 (raiz exata)

Raiz de 15
15 – 1 = 14 – 3 = 11 – 5 = 6 – 7 = -1
     ^        ^        ^       ^ = 4 (raiz está entre 3 e 4)
```

Exercício 20

`neperiano()`

Na matemática, o número neperiano (também conhecido como número de Euler, número de Napier, constante de Néper, constante matemática e, número exponencial) é a base dos logaritmos naturais. O número neperiano é definido pelo seguinte limite e vale aproximadamente 2,718.281.828.459.045.235.360.287.

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

O número também pode ser escrito como a soma da série infinita:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

Considerando que a fórmula da série infinita acima apresenta cinco termos visíveis (1/4! é o quinto), calcule e retorne o número neperiano para *n* termos.

Exercício 21

`neperianoEficiente()`

Caso, na resolução do exercício anterior, tenham sido utilizados comandos de iteração aninhados, tente resolvê-lo com um único comando de iteração.

Exercício 22

`isPerfeito()`

Retorne se *n* é um número perfeito, reduzido ou abundante.

Dado *n*, inteiro e positivo, diz-se que *n* é perfeito se for igual à soma de seus divisores positivos diferentes dele mesmo. Assim, 6 é um número perfeito, pois 1 + 2 + 3 =6. Contudo, números perfeitos são bastante raros: o quinto número perfeito é 33.550.336.

No século I d.C., havia, além da divisão dos números perfeitos, os números abundantes e os reduzidos:

- abundantes: se o número é inferior à soma dos seus divisores exceto ele próprio;
  Por exemplo, 12: 1 + 2 + 3 + 4 + 6 = 16.
- reduzidos: se o número é superior à soma dos seus divisores exceto ele próprio;
  Por exemplo, 9: 1 + 3 = 4.

Isto posto, os retornos possíveis são -1, 0 e 1, classificando *n* como, respectivamente, reduzido, perfeito ou abundante.

Exercício 23

`tresN()`

Suponha um número *n* qualquer, se *n* é par então *n* agora é *n*/2, se *n* é ímpar, *n* agora é 3*n+1. Assim para *n* = 3, calculamos a seguinte tabela:

```
 3 ► 10          4 ► 2
10 ►  5          2 ► 1
 5 ► 16          1 ► 4
16 ►  8          4 ► 2
 8 ►  4          2 ► 1
```

Observe que a partir da sétima iteração, a sequência 4 2 1 começa a se repetir. Calcule e retorne, para *n*, o número de iterações para se chegar ao primeiro 1.

Exercício 24

`sen()`

Calcule e retorne o *sen*(*n*), considerando que *n* é um ângulo representado em radianos. O valor do seno de *n* será calculado pela soma dos 5 primeiros termos da série a seguir:

```
sen n = n − (n^3)/3! + (n^5)/5! − (n^7)/7! + ...
```