INF 112, Aula prática 11 – Operadores e Exceções – 2018/2

1. O objetivo dessa questão é implementar uma classe que represente, simbolicamente, um número racional. Por definição, um número racional, x, sempre pode ser escrito como o quociente de dois números inteiros p e q ($q \neq 0$), i.e., $x = \frac{p}{q}$.

   O código abaixo mostra o esqueleto (com construtores e função de impressão) de como um número racional deve ser representado. Veja que o vetor `x` deve guardar em sua primeira componente o numerador (p) e em sua segunda componente o denominador (q).

```
#include <iostream>
using namespace std;
class Racional {
    private:
        int *x;
    public:
        Racional() {x = new int[2];x[0] = 0; x[1] = 1;}
        Racional(int p, int q) {
            x = new int[2];
            x[0] = p;
            x[1] = q;
        }
        void imprime(){cout <<  "(" << x[0] << "," << x[1] << ")" << endl;}
};
```

   Estenda a classe acima para que a função `main` que segue funcione em conformidade com as operações de soma, subtração, multiplicação e divisão no conjunto dos números racionais. Além disso, sua implementação não deve permitir vazamentos de memória, referências danosas, etc. Você pode modificar a implementação das funções dadas, mas a representação do número racional (vetor dinâmico de duas componentes) deve ser mantida.

   IMPORTANTE:

   1. Um número racional negativo deve sempre ter o numerador negativo e o denominador positivo. Um número racional positivo deve sempre ter numerador e denominador positivos;
   2. Os números racionais devem sempre estar em sua forma reduzida (e.g. $\frac{10}{20}$ tem forma reduzida $\frac{1}{2}$). Para essa tarefa, você precisará escrever uma função que calcule o máximo divisor comum entre dois inteiros. Qualquer implementação correta dessa função, mesmo que não eficiente, será aceita (desde que seja razoável). No entanto, pesquise sobre o algoritmo de Euclides.
   3. Você deve tratar todas as exceções. Por exemplo, se a alocação de memória no construtor falhar, seu programa deve lançar `MemoriaExcept` e se em algum ponto do seu código houver uma divisão por zero, seu programa deve lançar `DivisaoPorZeroExcept`. As exceções devem ter mensagens informativas, passadas como argumentos de seus construtores.

   Além dos operadores básicos, você deve implementar a operação de raiz quadrada (`sqrt`) racional. Nesse caso, $\sqrt{\frac{p}{q}} = \frac{\sqrt{p}}{\sqrt{q}}$ (com $q \neq 0$), onde a raiz quadrada de um inteiro a é o maior número inteiro, b, tal que $b^2 \leq a$. Para a implementação dessa função, você não pode usar nenhuma operação de raiz quadrada já implementada. Dica: *busca binária*. Se o argumento da operação for negativo, sua função deve lançar `RaizDeNumeroNegativoExcept`.

Implemente a operação `lg`, logaritmo na base 2 de um número racional. Proponha como essa operação deva funcionar, seguindo o mesmo molde da operação `sqrt`. Se o argumento da função não for positivo, deve-se lançar `LogDeNaoPositivoExcept`.

Por fim, estenda a `main` para testar suas novas operações e exceções.

```
// Você NÃO deve modificar o código abaixo
Racional f(Racional x, Racional y) {
    return x + (x / y);
}

int main (int argc, char *argv[]) {
    Racional x(atoi(argv[1]), atoi(argv[2])); // usa argumentos da linha de comando
    Racional y(atoi(argv[3]), atoi(argv[4])); // usa argumentos da linha de comando
    x.imprime();
    Racional z = x + y;
    z.imprime();
    z = x - y;
    z.imprime();
    z = x + 1;
    z.imprime();
    z = f(x, y);
    z.imprime();
    return 0;
}
```

Exemplo de entrada e saída:

```
g++ -o 1 1.cpp

./1 10 -20 30 40
(-1,2)
(1,4)
(-5,4)
(1,2)
(-7,6)
```

Explicação: $x = \frac{10}{-20} = \frac{-1}{2} = (-1, 2)$; $y = \frac{30}{40} = \frac{3}{4} = (3, 4)$; $x + y = \frac{1}{4} = (1, 4)$; $x - y = \frac{-5}{4} = (-5, 4)$; $x + 1 = \frac{1}{2} = (1, 2)$; $x + \frac{x}{y} = \frac{-7}{6} = (-7, 6)$.