2º Laboratório ECOP13 - Classes – 31 de março 2023

1ª Questão: Defina uma classe que represente um retangulo, com os atributos comprimento e largura. Nomeie a classe como **CRetangulo**. Setar o valor padrão desses atributos para 1. Criar funções de acesso para cada um dos atributos, validando os valores como números entre 1 e 20. Definir construtores que permitam o recebimento do valor de um atributo como parâmetro. Criar métodos para o calculo da *area* e *perímetro* do retangulo. Criar uma função capaz de imprimir esse retangulo conforme descrito na questão 5 do lab 1. Criar um método para verificar se o retangulo é um quadrado.

2ª Questão: Alterar a classe **CRetangulo** da questão 1 para permitir que o usuário de um programa que a utilize consiga visualizar o momento da criação e da destruição de cada um dos objetos instanciados.

3ª Questão: Alterar a classe **CRetangulo** para separar a declaração da implementação da classe, ressaltando o uso do operador de qualificação de escopo (::), precedendo o nome de um método na classe no local de sua implementação.

4ª Questão: Acrescente uma função a classe **CRetangulo** que permita que o usuário entre com os atributos do retângulo.

5ª Questão: Utilizar a classe CRetangulo para criar um vetor de 5 objetos e permitir que o usuário entre com os atributos de cada um deles. Acrescente no final do programa a impressão de cada um deles.

6ª Questão: Criar um programa que utilize um objeto da classe CRetangulo através de um ponteiro. Observar o uso do operador -> para acessar um membro publico do objeto.

7ª Questão: Escreva uma classe que represente **polígonos regulares**. O construtor deve receber o número de lados e o comprimento de cada lado. Acrescente um método `area`(), que deve calcular a área dos polígonos com a fórmula $\frac{1}{4} nb^2 \frac{\cos(\pi / n)}{\text{sen}(\pi / n)}$, e um método `perimetro`(). Onde **n** representa o número de lados e **b** o comprimento de cada lado. Escreva também um método que imprima o nome do polígono baseado no seu número de lados. (Simplificação: considere polígonos de 3 até 10 lados.

8ª Questão: Dada a classe que representa uma fração, criar um programa para testar todas as suas funcionalidades.

```
// arquivo CFracao.h - interface para a classe CFracao
//
#ifndef ID_CFRACAO
#define ID_CFRACAO

class CFracao
{
protected:
      int m_numerador;
```

```
    int m_denominador;

    // responde ao receptor com o mínimo denominador comun
    CFracao Reduzida(void);

public:
    // Construtor sem parametros inline
    CFracao(void){
        m_numerador = 1;
        m_denominador = 1;
    }
    CFracao(int Num, int Denom) : m_numerador(Num),
                                  m_denominador(Denom) { };
    CFracao( const CFracao& f) // Construtor de copia
    {
        m_numerador = f.m_numerador;
        m_denominador = f.m_denominador;
    }
    ~CFracao(void){ };          // Destrutor

    //
    //métodos de acesso
    //
    int getNumerador(void) { return m_numerador; }
    int getDenominador(void) { return m_denominador; }

    //
    //métodos aritiméticos
    //
    // retorna uma nova Fracao que é a soma do receptor com _F
    CFracao Somar(CFracao _F);
    // retorna uma nova Fracao que é a subtração do receptor com _F
    CFracao Subtrair(CFracao _F);
    // retorna uma nova Fracao que o produto do receptor e _F
    CFracao Multiplicar(CFracao _F);
    // retorna uma nova Fracao que o quociente do receptor e _F
    CFracao Dividir(CFracao _F);

    //
    //métodos de coparação
    //
    // devolve verdadeiro se receptor menor que _Fracao
    int MenorQue(CFracao _Fracao);
    // devolve verdadeiro se receptor maior que _Fracao
    int MaiorQue(CFracao _Fracao);
    // devolve verdadeiro se receptor igual a _Fracao
    int Igual(CFracao _Fracao);

    //
    //métodos de conversão
    //

    // devolve o valor da fração como float
    float ComoFloat(void);

    //
    //métodos de impressão
    //
```

```
      // mostrar o receptor no formato m_numerador/m_denominador
      void Print(void);

};

#endif // ID_CFRACAO
```

```
//Arquivo  CFracao.cpp - Implementação da classe CFracao

#include "CFracao.h"
#include <iostream>
using namespace std;

//
// Métodos Protegidos da classe CFracao
//
CFracao CFracao::Reduzida(void)
{
      int gcd = 1;
      int minimo = m_numerador;
      if (m_numerador > m_denominador)
            minimo = m_denominador;
      for(int i = 1; i <= minimo; i++)
      {
            if ((m_numerador%i == 0) && (m_denominador%i == 0))
                  gcd = i;
      }
      m_numerador /= gcd;
      m_denominador /= gcd;
      return (*this);
}

//
// Métodos Aritméticos da classe CFracao
//
// retorna uma nova Fracao que é a soma do receptor com _Fracao
CFracao CFracao::Somar(CFracao _Fracao)
{
      CFracao temp(m_numerador*_Fracao.m_denominador +
   m_denominador*_Fracao.m_numerador, m_denominador*_Fracao.m_denominador
);
      return temp.Reduzida();
}

// retorna uma nova Fracao que é a subtração do receptor com _Fracao
CFracao CFracao::Subtrair(CFracao _Fracao)
{
      CFracao temp(m_numerador*_Fracao.m_denominador -
   m_denominador*_Fracao.m_numerador, m_denominador*_Fracao.m_denominador
);
      return temp.Reduzida();
}

// retorna uma nova Fracao que o produto do receptor e _Fracao
CFracao CFracao::Multiplicar(CFracao _Fracao)
```

```
{
      CFracao temp(m_numerador*_Fracao.m_numerador,
m_denominador*_Fracao.m_denominador );
      return temp.Reduzida();
}

// retorna uma nova Fracao que o quociente do receptor e _Fracao
CFracao CFracao::Dividir(CFracao _Fracao)
{
      CFracao temp(m_numerador*_Fracao.m_denominador,
                  m_denominador*_Fracao.m_numerador );
      return temp.Reduzida();
}

//
// Métodos de comparação da classe CFracao
//
// devolve verdadeiro se receptor menor que _Fracao
int CFracao::MenorQue(CFracao _Fracao)
{
      return (m_numerador*_Fracao.m_denominador <
m_denominador*_Fracao.m_numerador );
}

// devolve verdadeiro se receptor maior que _Fracao
int CFracao::MaiorQue(CFracao _Fracao)
{
      return (m_numerador*_Fracao.m_denominador >
m_denominador*_Fracao.m_numerador );
}

// devolve verdadeiro se receptor igual a _Fracao
int CFracao::Igual(CFracao _Fracao)
{
      return (m_numerador*_Fracao.m_denominador ==
m_denominador*_Fracao.m_numerador );
}

//
// Métodos de conversão
//
// devolve o valor da fração como float
float CFracao::ComoFloat(void)
{
      return ((float)m_numerador/(float)m_denominador);
}

//
// Métodos de impressão
//
// mostrar o receptor no formato m_numerador/m_denominador
void CFracao::Print(void)
{
      cout << m_numerador << "/" << m_denominador;
}
```