



ECOP13A-Lab3

Guia de Laboratório

Prof. André Bernardi
andrebernardi@unifei.edu.br

16 de abril-2025



3º Laboratório ECOP13A

16 Abril 2025



1ª Questão: Defina uma classe que represente um retângulo, 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 cálculo da *area* e *perímetro* do retângulo. Criar uma função capaz de imprimir esse retângulo conforme descrito na questão 5 do lab1. Criar um método para verificar se o retângulo 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 público do objeto.

1ª Questão



1ª: Criar uma classe que represente um retângulo com as seguintes características:

- Nome: **CRetangulo**, com os atributos comprimento e largura. 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 cálculo da *área* e *perímetro* do retângulo.
- Incorporar a função de impressão de retângulo conforme descrita na questão 5 do lab1. Criar os atributos necessários na classe para que ela funcione corretamente.
- Criar um método para verificar se o retângulo é um quadrado.

1ª questão – Exemplo de Solução



```
1 // Exercício 1 - arquivo retangulo.h
2 #include <iostream>
3 using namespace std;

#ifndef RETANGULO_H
#define RETANGULO_H

class CRetangulo
{
private:
    int largura, comprimento;
public:
    CRetangulo();
    CRetangulo(int c, int l);
    ~CRetangulo(){
        cout << endl;
        cout << " Destruindo retângulo..." << endl;
    }
    int getLarg();
    int getComp();
    void setLarg(int l);
    void setComp(int c);
    bool is_valid(int c, int l);

    int area();
    int perimetro();
    bool isQuadrado();
    void imprime(char b, char p);

    void leitura();
    void imprime();
};

#endif // RETANGULO_H
```



Construtores

```
#include <iostream>
#include "retangulo.h"

// Construtor padrão
CRetangulo::CRetangulo(){
    largura = 1;
    comprimento = 1;
    cout << endl;
    cout << "Foi criado um obj padrao!" << endl; // ( 2 )
}

// Construtor personalizado
CRetangulo::CRetangulo(int c, int l){
    setLarg(1);
    setComp(c);

    cout << endl;
    cout << "Foi criado um obj personalizado!" << endl;
}
```



Métodos de acesso

```
// recupera valores do objeto
int CRetangulo::getLarg() {
    return largura;
}
int CRetangulo::getComp() {
    return comprimento;
}

// Ajusta valores do objeto
void CRetangulo::setComp(int c) {
    if(c >= 1 && c <= 20)
        comprimento = c;
    else comprimento = 1;
}
void CRetangulo::setLarg(int l) {
    if(l >= 1 && l <= 20)
        largura = l;
    else largura = 1;
}
```



Verificações e cálculos

```
// Validação dos atributos
bool CRetangulo::is_valid(int c, int l){
    return (c>=1 && c<=20 && l>=1 && c<=20);
}

// Calculos
int CRetangulo::area(){
    return largura*comprimento;
}

int CRetangulo::perimetro(){
    return 2*(largura + comprimento);
}

bool CRetangulo::isSquare(){
    return (largura == comprimento);
}
```




Impressão e Leitura

```
// IMPRIMIR - LEITURA
void CRetangulo::imprime(){
    cout << "largura: " << largura << " comprimento: " << comprimento << endl;
}

void CRetangulo::leitura(){
    int c, l;
    cout << "Digite um comp: "; cin >> c;
    cout << "Digite uma larg: "; cin >> l;

    /*
    while(!is_valid(c, l)){
        cout << "Digite novamente um comp: "; cin >> c;
        cout << "Digite novamente uma larg: "; cin >> l;
    }
    */
    setLarg(l);
    setComp(c);
}
```




Impressão ASCII

```
//impressão ASCII
void CRetangulo::imprime(char b, char p){
    int lin = largura;
    int col = comprimento;

    for(int i=0; i<lin; i++){
        for(int j=0; j<col; j++){
            if(i == 0 || j == 0 || i == lin-1 || j == col-1)
                cout << b;
            else cout << p;
        }
        cout << endl;
    }
}
```

2ª Questão



2ª: Alterar a classe ***CRetangulo*** da 1ª questão 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. Para isso basta inserir linhas de impressão nos construtores e no destrutor.



2ª questão – Exemplo de Solução

```
#include <iostream>
#include "retangulo.h"

// Construtor padrão
CRetangulo::CRetangulo() {
    largura = 1;
    comprimento = 1;
    cout << endl;
    cout << "Foi criado um obj padrao!" << endl; // ( 2 )
}

// Construtor personalizado
CRetangulo::CRetangulo(int c, int l){
    setLarg(l);
    setComp(c);

    cout << endl;
    cout << "Foi criado um obj personalizado!" << endl;
}
```

```
~CRetangulo() {
    cout << endl;
    cout << " Destruindo retângulo..." << endl;
}
```

3ª Questão



3ª: Alterar a classe ***CRetangulo*** da 1ª questão, caso ainda não tenha feito dessa forma, separando a declaração da classe em um arquivo .h e sua implementação em um arquivo .cpp, lembrando de utilizar corretamente o operador de qualificação de escopo (::), precedendo o nome de um método da classe no local de sua implementação.

4ª Questão



4ª: Acrescente uma função membro a classe **CRetangulo** que permita que o usuário entre com todos os atributos do retângulo. De o nome de **leitura** a esse método.

```
void CRetangulo::leitura() {
    int c, l;
    cout << "Digite um comp: "; cin >> c;
    cout << "Digite uma larg: "; cin >> l;

    /*
    while(!is_valid(c, l)){
        cout << "Digite novamente um comp: "; cin >> c;
        cout << "Digite novamente uma larg: "; cin >> l;
    }
    */
    setLarg(l);
    setComp(c);
}
```

5ª e 6ª Questão



- 5ª: 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: Criar um programa que utilize um objeto da classe **CRetangulo** através de um ponteiro. Observar o uso do operador -> para acessar um membro público do objeto.



5ª e 6ª questão

Exemplo de Solução

```
CRetangulo ret3[5]; // ( 5 )
```

```
cout << endl << endl << "Leitura vetor obj" << endl;  
for(int i=0; i<4; i++){  
    ret3[i].leitura();  
}
```

```
cout << endl << endl << "Impressão dos dados" << endl;  
for(int i=0; i<4; i++){  
    ret3[i].imprime();  
}
```

```
// ( 6 )
```

```
cout << endl << "Ret4 com ponteiro" << endl;
```

```
CRetangulo *ret4 = new CRetangulo;
```

```
ret4->leitura();  
ret4->imprime();
```

```
delete ret4;
```


7a Questão



7ª: Escreva uma classe que represente polígonos regulares. O construtor deve receber o número de lados **n** e o comprimento de cada lado **b**. Acrescente um método `area()`, que deve calcular a área dos polígonos com a fórmula:

$$\frac{1}{4}nb^2 \frac{\cos(\pi / n)}{\sin(\pi / n)}$$

, e um método `perimetro()`. Escreva também um método que imprima o nome do polígono regular baseado no seu número de lados. (Simplificação: considere polígonos de 3 até 10 lados.

7ª questão

Exemplo de Solução



```
#include <string>

using namespace std;

#ifndef POLIGONOS_H
#define POLIGONOS_H

class Poligonos{
private:
    int n, b;
public:
    //Poligonos();
    Poligonos(int n = 1, int b = 1);

    ~Poligonos(){ }

    float Area();
    float Perimetro();
    void preenche(int x, int y);
    string Nome();
};
#endif
```

7ª questão

Exemplo de Solução



```
#include <iostream>
#include "poligonos.h"
#define PI 3.141592

using namespace std;

void Poligonos::preenche(int x, int y){
    this->n = x;
    this->b = y;
}
Poligonos::Poligonos(int n, int b){
    this->n = n;
    this->b = b;
}
float Poligonos::Area(){
    return (0.25)*n*b*b* ( cos(PI/n) / sin(PI/n) );
}
float Poligonos::Perimetro(){
    return n*b;
}
```

7ª questão

Exemplo de Solução



```
string Poligonos::Nome() {  
    if(n == 10)  
        return "Decagono";  
    else if(n == 9)  
        return "Eneagono";  
    else if(n == 8)  
        return "Octogono";  
    else if(n == 7)  
        return "Heptagono";  
    else if(n == 6)  
        return "Hexagono";  
    else if(n == 5)  
        return "Pentagono";  
    else if(n == 4)  
        return "Quadrado";  
    else if(n == 3)  
        return "Triangulo";  
  
    else return "Nao Identificado!";  
}
```

7ª questão

Exemplo de Solução



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

int main(){

    // ( 7 )
    Poligonos poli;

    int n, b;
    cout << "Enter with N and B: ";
    cin >> n >> b;
    poli.preenche(n, b);

    cout << "Nome = " << poli.Nome() << endl;
    cout << "Area = " << poli.Area() << endl;
    cout << "Perimetro = " << poli.Perimetro() << endl;
}
```

8ª Questão



8ª: 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
{
    private:
        int m_numerador;
        int m_denominador;

        // reduz a fração em sua forma mínima
        CFracao Reduzida(void);
};
```

```
public:
```

```
    CFracao(void) {                // Construtor sem parâmetros inline  
        m_numerador = 1;  
        m_denominador = 1;  
    }
```

```
    // Construtor com parâmetros inline
```

```
    CFracao(int Num, int Denom) : m_numerador(Num),  
                                   m_denominador(Denom) { };
```

```
    CFracao(const CFracao& f) // Construtor de copia inline  
    {
```

```
        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 aritméticos
```

```
    // retorna uma nova fração que é a soma do receptor com _F
```

```
    CFracao Somar(CFracao _F);
```

```
    // retorna uma nova fração 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 comparação
// 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 Privativos 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;
}
```



8ª questão

Exemplo de Solução



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

int main(){
    // ( 8 )
    CFracao f1(8, 4);

    cout << "Numerador F1 = " << f1.getNumerador() << endl;
    cout << "Denominador F1 = " << f1.getDenominador() << endl <<
endl;

    CFracao f2(4, 2);

    cout << "Numerador F2= " << f2.getNumerador() << endl;
    cout << "Denominador F2= " << f2.getDenominador() << endl <<
endl;
```

```
CFracao f3 = f1.Somar(f2);  
cout << "Soma = "; f3.Print(); cout << endl;  
f3 = f1.Subtrair(f2);  
cout << "Subtrair = "; f3.Print(); cout << endl;  
f3 = f1.Multiplicar(f2);  
cout << "Mult = "; f3.Print(); cout << endl;  
f3 = f1.Dividir(f2);  
cout << "Div = "; f3.Print(); cout << endl;
```

```
if(f1.MenorQue(f2)) {  
    cout << "Menor = "; f1.Print(); cout << endl;  
}else{  
    cout << "Menor = "; f2.Print(); cout << endl;  
}  
if(f1.MaiorQue(f2)) {  
    cout << "Maior = "; f1.Print(); cout << endl << endl;  
}else{  
    cout << "Maior = "; f2.Print(); cout << endl << endl;  
}  
if(f1.Igual(f2)) {  
    cout << "Iguais" << endl;  
}else cout << "Not iguais" << endl;
```

```
cout << "Float = " << f3.ComoFloat();  
f3.Print();
```

```
return 0;
```

