ECOP13A-Lab5

Guia de Laboratório

ECOP13A - Programação Orientada a Objetos
Prof. André Bernardi
andrebernardi@unifei.edu.br

Universidade Federal de Itajubá

5º Laboratório ECOP13A
09 de maio 2025

1ª Questão

1ª: Alterar a classe **CFracao** do laboratório 3 de modo a utilizar sobrecarga de operadores para as funções de soma, subtração, multiplicação e divisão. Em seguida acrescente os seguintes itens a classe:

a. Operadores para permitir a comparação de frações (<, >, <=, >=, ==, !=).

b. Operadores para realizar a impressão e a leitura das frações (<< e >>).

c. Criar um programa que teste TODAS funcionalidades implementadas nos itens acima.

1ª questão – Exemplo de Solução

```
// Lab 04 - Exercício 1 – arquivo CFracao.h

#include <iostream>
using namespace std;

#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);
    operator float() const;

    //
    // ---- métodos de impressão
    // mostrar o receptor no formato m_numerador/m_denominador
    void Print(void) const;

    // ---- declaração dos operadores aritméticos
    CFracao operator + (const CFracao&);
    CFracao operator - (const CFracao&);
    CFracao operator * (const CFracao&);
    CFracao operator / (const CFracao&);

    // ---- declaração dos operadores de comparação
    bool operator < (const CFracao& );
    bool operator > (const CFracao& );
    bool operator <= (const CFracao& );
    bool operator >= (const CFracao& );
    bool operator == (const CFracao& );
    bool operator != (const CFracao& );

    //operadores leitura e escrita
    friend ostream& operator << ( ostream& , const CFracao& );
    friend istream& operator >> ( istream& , CFracao& );
};

#endif // ID_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 _F)
{
    return (m_numerador*_F.m_denominador > m_denominador*_F.m_numerador );
}

// devolve verdadeiro se receptor igual a _Fracao
int CFracao::Igual(CFracao _F)
{
    return (m_numerador*_F.m_denominador == m_denominador*_F.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) const
{
    cout << m_numerador << "/" << m_denominador;
}
```

```
// ---- Operadores Aritméticos
CFracao CFracao::operator+(const CFracao & _Fracao)
{
     CFracao temp(m_numerador*_Fracao.m_denominador + m_denominador*_Fracao.m_numerador,
                  m_denominador*_Fracao.m_denominador);
     return temp.Reduzida();
}

CFracao CFracao::operator-(const CFracao & _Fracao)
{
     CFracao temp(m_numerador*_Fracao.m_denominador - m_denominador*_Fracao.m_numerador,
                  m_denominador*_Fracao.m_denominador);
     return temp.Reduzida();
}

CFracao CFracao::operator*(const CFracao & _Fracao)
{
     CFracao temp(m_numerador*_Fracao.m_numerador, m_denominador*_Fracao.m_denominador);
     return temp.Reduzida();
}

CFracao CFracao::operator/(const CFracao & _Fracao)
{
     CFracao temp(m_numerador*_Fracao.m_denominador, m_denominador*_Fracao.m_numerador);
     return temp.Reduzida();
}
```

```
// ---- Operadores de comparação
bool CFracao::operator<(const CFracao & _Fracao)
{
    return ((float)(*this) < (float)_Fracao);
}

bool CFracao::operator>(const CFracao & _Fracao)
{
    return ((float)(*this) > (float)_Fracao);
}

bool CFracao::operator<=(const CFracao & _Fracao)
{
    return ((float)(*this) <= (float)_Fracao);
}

bool CFracao::operator>=(const CFracao & _Fracao)
{
    return ((float)(*this) >= (float)_Fracao);
}

bool CFracao::operator==(const CFracao & _Fracao)
{
     return ((float)(*this) == (float)_Fracao);
}

bool CFracao::operator!=(const CFracao & _Fracao)
{
     return ((float)(*this) != (float)_Fracao);
}
```

```
// ---- Operadores de Conversão
CFracao::operator float (void) const
{
     return ((float)m_numerador/(float)m_denominador);
}

// ---- Operadores entrada e saída
ostream& operator << (ostream& output, const CFracao& fracao)
{
     //output << fracao.m_numerador << "/" << fracao.m_denominador;
    fracao.Print();
     return output;
}

istream& operator >> (istream& input, CFracao& fracao)
{
    input >> fracao.m_numerador >> fracao.m_denominador;
    return input;
}
```

```cpp
#include <iostream>
#include "CFracao.h"

using namespace std;

int main()
{
    CFracao a, b;

    cout << "Fracao A: ";
    cin >> a;
    cout << "Fracao B: ";
    cin >> b;

    cout << "Fracao A + B: " << a + b << endl;
    cout << "Fracao A - B: " << a - b << endl;
    cout << "Fracao A * B: " << a * b << endl;
    cout << "Fracao A / B: " << a / b << endl;

    if(a < b) cout << a << " < " << b << endl;
    if(a <= b) cout << a << " <= " << b << endl;
    if(a > b) cout << a << " > " << b << endl;
    if(a >= b) cout << a << " >= " << b << endl;
    if(a != b) cout << a << " != " << b << endl;
    if(a == b) cout << a << " = " << b << endl;
}
```

2ª Questão

2ª : Alterar a classe Complexo do laboratório 4 de modo a utilizar sobrecarga de operadores para as funções de soma, subtração, multiplicação e divisão. Em seguida acrescente os seguintes itens a classe:

a) Operadores para realizar a impressão e a leitura dos Complexos (<< e >>).

b) Implementar os operadores == e !=

c) Criar um programa que teste TODAS funcionalidades implementadas nos itens acima.

2ª questão – Exemplo de Solução

```h
// Exercício 2 – arquivo complexo.h
#ifndef COMPLEXO_H
#define COMPLEXO_H

class Complexo
{
    private:
        double real, imag;
        static int n;

    public:
        Complexo() {
            real = 1;
            imag = 1;
            n++;
        }
        Complexo(double a, double b) {
            real = a;
            imag = b;
            n++;
        }
        Complexo(const Complexo& c) {
            real = c.real;
            imag = c.imag;
            n++;
        }
        ~Complexo() {n--;}

        Complexo operator + (Complexo);
        Complexo operator - (Complexo);
        Complexo operator * (Complexo);
        Complexo operator / (Complexo);

        bool operator == (Complexo);
        bool operator != (Complexo);

        Complexo somar(Complexo);
        Complexo subtrair(Complexo);
        Complexo multiplicar(Complexo);
        Complexo dividir(Complexo);

        void setReal(double a) {real = a;}
        void setImaginario(double a){imag=a;}
        double getReal() {return real;}
        double getImaginario() {return imag;}
        int getObjetos() {return n;}

        double modulo();
        void print();
    friend ostream& operator<<(ostream&, const Complexo&);
     friend istream& operator>>(istream&, Complexo&);
};

#endif
```

```cpp
#include <iostream>
#include <cmath>
#include "complexo.h"

using namespace std;

// inicialização do membro estático
int Complexo::n = 0;

//função somar - recebe um complexo como parâmetro e retorna um complexo
Complexo Complexo::somar(Complexo _complexo)
{
    Complexo temp(getReal() + _complexo.getReal(),
                  getImaginario() + _complexo.getImaginario());
    return temp;
}
//função subtrair - recebe um complexo como parâmetro e retorna um complexo
Complexo Complexo::subtrair(Complexo _complexo)
{
    Complexo temp(getReal() - _complexo.getReal(),
                    getImaginario() - _complexo.getImaginario());
    return temp;
}
//função multiplicar - recebe um complexo como parâmetro e retorna um complexo
Complexo Complexo::multiplicar(Complexo _complexo)
{
    Complexo temp(
        getReal()*_complexo.getReal() - getImaginario()*_complexo.getImaginario(),
        getReal()*_complexo.getImaginario() + getImaginario()*_complexo.getReal());
    return temp;
}
```


```cpp
//função dividir - recebe um complexo como parametro e retorna um complexo
Complexo Complexo::dividir(Complexo _c)
{
    double a = ( real *_c.real + imag *_c.imag ) /
                (pow(_c.real, 2) + pow(_c.imag, 2));
    double b = (_c.real * imag - real *_c.imag ) /
                (pow(_c.real, 2) + pow(_c.imag, 2));
    Complexo temp(a, b);
    return temp;
}

//função modulo - calcula o modulo do complexo
double Complexo::modulo()
{
    return sqrt(pow(getReal(), 2) + pow(getImaginario(), 2));
}

//função print - imprime um complexo com o formato desejado
void Complexo::print()
{
    cout << getReal() << " ";
    if(getImaginario() < 0) cout << getImaginario() << "i" << endl;
    else cout << "+" << getImaginario() << "i" << endl;
}
```

```
Complexo Complexo::operator + (Complexo _complexo)
{
    Complexo temp(getReal() + _complexo.getReal(),
                  getImaginario() + _complexo.getImaginario());
    return temp;
}

Complexo Complexo::operator - (Complexo _complexo)
{
    Complexo temp(getReal() - _complexo.getReal(),
                  getImaginario() - _complexo.getImaginario());
    return temp;
}

Complexo Complexo::operator * (Complexo _complexo)
{
    Complexo temp(getReal()*_complexo.getReal() - getImaginario()*_complexo.getImaginario(),
     getReal()*_complexo.getImaginario() + getImaginario()*_complexo.getReal());
    return temp;
}

Complexo Complexo::operator / (Complexo _complexo)
{
    double a = (getReal()*_complexo.getReal() + getImaginario()*_complexo.getImaginario()) /
(pow(_complexo.getReal(), 2) + pow(_complexo.getImaginario(), 2));
    double b = (_complexo.getReal()*getImaginario() - getReal()*_complexo.getImaginario()) /
(pow(_complexo.getReal(), 2) + pow(_complexo.getImaginario(), 2));
    Complexo temp(a, b);
    return temp;
}
```

```
bool Complexo::operator == (Complexo _complexo)
{
    return ( (real ==_complexo.real) && (imag == _complexo.imag)) ;
}

bool Complexo::operator != (Complexo _complexo)
{
    return !((*this) == _complexo);
}

ostream& operator << (ostream& output, const Complexo& complexo)
{
     output << complexo.real << " ";
    if(complexo.imag < 0) output << complexo.imag << "i";
    else output << "+" << complexo.imag << "i";
     return output;
}

istream& operator >> (istream& input, Complexo& complexo)
{
    input >> complexo.real >> complexo.imag;
    return input;
}
```

Main

```
#include <iostream>
#include <cmath>
#include "complexo.h"

using namespace std;

int main()
{
   Complexo a(1,2);
   Complexo b(3,4);
   Complexo aux;
   int ans;

   cout << "A: ";
   a.print();
   cout << "B: ";
   b.print();

   cout << "A + B: ";
   aux = a.somar(b);
   aux.print();

   cout << "A - B: ";
   aux = a.subtrair(b);
   aux.print();

   cout << "A * B: ";
   aux = a.multiplicar(b);
   aux.print();

   cout << "A / B: ";
   aux = a.dividir(b);
   aux.print();

   cout << "|A|: " << a.modulo() << endl;
   cout << "|B|: " << b.modulo() << endl;

   cout << "Qt de objetos: " <<
           a.getObjetos() << endl;
```

Main


```
///Operadores

   cout << "Digite os valores de A e B: ";
   cin >> a >> b;
   cout << "A: " << a << " // B: " << b << endl;

   cout << "A + B: " << a + b << endl;
   cout << "A - B: " << a - b << endl;
   cout << "A * B: " << a * b << endl;
   cout << "A / B: " << a / b << endl;

    if(a != b) cout << a << " != " << b << endl;
    if(a == b) cout << a << " = " << b << endl;

}
```

3ª Questão

- **3ª:** Alterar a classe que representa um número inteiro longo com 30 dígitos, do laboratório 4 para fazer uso da sobrecarga de operadores.
  a) Acrescente operadores para permitir a comparação (<, >, <=, >=, ==, !=).
  b) Operadores para realizar a impressão e a leitura (<< e >>).
  c) Criar um programa que teste TODAS funcionalidades implementadas nos itens acima.

3ª questão
Exemplo de Solução


```
#include <iostream>
using namespace std;

#ifndef BIGINT_H
#define BIGINT_H

class BigInt
{
   private:
      int num[31];
      int len;

      void inicializar()
      {
         for(int i = 0; i < 31; i++) num[i] = 0;
      }

   public:
      BigInt() {inicializar();}
      ~BigInt() {};

      void leia();
      void print();
      BigInt soma(BigInt);
      BigInt subtrai(BigInt);
```

3ª questão
Exemplo de Solução

```
        // operadores
        friend ostream& operator<<(ostream&, const BigInt&);
           friend istream& operator>>(istream&, BigInt&);
        BigInt operator + (BigInt);
        BigInt operator - (BigInt);

        bool operator < (BigInt);
        bool operator > (BigInt);
        bool operator <= (BigInt);
        bool operator >= (BigInt);
        bool operator == (BigInt);
        bool operator != (BigInt);

};
#endif
```

3ª questão
Exemplo de Solução


```cpp
#include <iostream>
#include <string>
#include "bigint.h"

using namespace std;

void BigInt::leia()
{
   string a;
   cin >> a;
   len = a.length();
   for(int i = 0; i < len; i++)
   {
      num[i] = a[len-i-1] - '0';
   }
}

void BigInt::print()
{
   for(int i = len-1; i >= 0; i--)
      cout << num[i];
   cout << endl;
}
```

SOMA


```
BigInt BigInt::soma(BigInt b)
{
   BigInt c;
   c.len = 0;

   for(int i = 0, vaiUm = 0; vaiUm || i < max(len, b.len); i++)
   {
      int x = vaiUm;
      if(i < len) x += num[i];
      if(i < b.len) x += b.num[i];
      c.num[c.len++] = x % 10;
      vaiUm = x / 10;
   }
   return c;
}
```

Subtração:


```
BigInt BigInt::subtrai(BigInt b)
{
   BigInt c;
   c.len = 0;

   for(int i = 0; i < max(len, b.len); i++)
   {
      int x = 0;
      if(i < len) x += num[i];
      if(i < b.len)
      {
         if(num[i] >= b.num[i])
            x -= b.num[i];
         else {
            x += 10 - b.num[i];
            num[i+1]--;
         }
      }
      c.num[c.len++] = x;
   }
   return c;
}
```

Operadores:

```
//operadores
istream& operator >> (istream& input, BigInt& numero)
{
     string a;
     input >> a;
     numero.len = a.length();
     for(int i = 0; i < numero.len; i++)
     {
        numero.num[i] = a[numero.len-i-1] - '0';
     }
     return input;
}

ostream& operator << (ostream& output, const BigInt& numero)
{
   for(int i = numero.len-1; i >= 0; i--)
      output << numero.num[i];
      return output;
}
```

```
BigInt BigInt::operator + (BigInt b)
{
   return this->soma( b );
}

BigInt BigInt::operator - (BigInt b)
{
   return this->subtrai(b);
}

bool BigInt::operator < (BigInt numero)
{
   if(len != numero.len){
      if(len > numero.len)
         return false;
      else
         return true;
   }
   else{
      for(int i = len-1; i >= 0; i--){
         if(num[i] < numero.num[i]) return true;
      }
   }
   return false;
}
```

```cpp
bool BigInt::operator > (BigInt numero)
{
   if(len != numero.len){
      if(len < numero.len)
            return false;
      else return true;
   } else {
      for(int i = len-1; i >= 0; i--)
         if(num[i] > numero.num[i])
            return true;
   }
   return false;
}

bool BigInt::operator == (BigInt numero)
{
   if(len == numero.len){
      for(int i = 0; i < len; i++)
         if(num[i] != numero.num[i]) return false;
      return true;
   }
   else return false;
}
```

```
bool BigInt::operator <= (BigInt numero)
{
   if(*this < numero || *this == numero)
         return true;
   else
         return false;
}

bool BigInt::operator >= (BigInt numero)
{
   if(*this > numero || *this == numero)
         return true;
   else
         return false;
}

bool BigInt::operator != (BigInt numero)
{
   return !(*this == numero);
}
```

Main

```
///Operadores

   cout << "Digite o valor de A: ";
   cin >> a;
   cout << "Digite o valor de B: ";
   cin >> b;
   cout << "A+B = " << a+b << endl;

   if(a >= b) cout << "A-B = " << a-b << endl;

   if(a < b) cout << a << " < " << b << endl;
   if(a > b) cout << a << " > " << b << endl;
   if(a <= b) cout << a << " <= " << b << endl;
   if(a >= b) cout << a << " >= " << b << endl;
   if(a == b) cout << a << " = " << b << endl;
   if(a != b) cout << a << " != " << b << endl;
```