

ECOP13A-Lab6

ECOP13A - Programação Orientada a Objetos

Prof. André Bernardi

andrebernardi@unifei.edu.br



Universidade Federal de Itajubá



6º Laboratório ECOP13A

16 de setembro 2024





1ª Questão

Declarar uma classe que represente um **polinômio** de ordem *n* do tipo:

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Sobrecarregue e implemente os operadores de Soma(+), Subtração(-), Leitura(>>) e Impressão(<<) para a classe declarada. Crie um programa para testar o uso dessa classe com suas funcionalidades.



// Lab 05 - Exercício 1 - arquivo polinomio.h



```
#ifndef POLINOMIO_H
#define POLINOMIO_H
```

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

```
class Polinomio{
    private:
        double *valores;
        int n;

    public:
        Polinomio();
        Polinomio(int);
        Polinomio(const Polinomio&);

        ~Polinomio();

        Polinomio operator = ( const Polinomio& );
        Polinomio operator+(Polinomio);
        Polinomio operator-(Polinomio);

        double& operator[](int);

        friend ostream& operator << (ostream&, Polinomio&);
        friend istream& operator >> (istream&, Polinomio&);
};
#endif
```

1ª questão – Exemplo de Solução

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

```
using namespace std;
```

```
Polinomio::Polinomio()
{
    n = 2;
    valores = new double[n];
    valores[0] = 1;
    valores[1] = 1;
}
```

```
Polinomio::Polinomio(int grau)
{
    n = grau + 1;
    valores = new double[n];
    for(int i = 0; i < n; i++)
        valores[i] = 1;
}
```

//construtor de copia é necessário pois a classe usa ptr

```
Polinomio::Polinomio(const Polinomio& p)
{
    n = p.n;
    valores = new double[n];
    for(int i = 0; i < p.n; i++)
    {
        valores[i] = p.valores[i];
    }
}
```



```
Polinomio::~~Polinomio()
```

```
{  
    delete[] valores;  
}
```

```
Polinomio Polinomio::operator=(const Polinomio& p)
```

```
{  
    delete [] valores; // limpar o ponteiro antigo  
    n = p.n;  
    valores = new double[n]; // alocar para o novo tamanho  
    for(int i = 0; i < p.n; i++)  
    {  
        valores[i] = p.valores[i]; //copiar valores  
    }  
    return *this;  
}
```

```
Polinomio Polinomio::operator+(Polinomio _pol)
```

```
{  
    Polinomio temp(max(_pol.n, n)-1);  
    int i;  
    for(i = 0; i < min(_pol.n, n); i++)  
        temp[i] = _pol.valores[i] + valores[i];  
  
    if(_pol.n > n)  
        for(int j = i; j < _pol.n; j++)  
            temp[j] = _pol.valores[j];  
    else  
        for(int j = i; j < n; j++)  
            temp[j] = valores[j];  
    return temp;  
}
```



```
Polinomio Polinomio::operator-(Polinomio _pol)
```

```
{
```

```
    Polinomio temp(max(_pol.n, n)-1);
```

```
    int i;
```

```
    for(i = 0; i < min(_pol.n, n); i++)
        temp[i] = valores[i] - _pol.valores[i];
```

```
    if(_pol.n > n)
        for(int j = i; j < _pol.n; j++)
            temp[j] = - _pol.valores[j];
```

```
    else
```

```
        for(int j = i; j < n; j++)
            temp[j] = valores[j];
```

```
    return temp;
```

```
}
```

```
istream& operator >> (istream& input, Polinomio& _pol)
```

```
{
```

```
    cout << "Polinomio: C0 + C1x1 + C2x2 + ... + Cnxn = 0" << endl;
```

```
    for(int i = 0; i < _pol.n; i++)
```

```
    {
```

```
        cout << "Digite o valor de C" << i << ": ";
```

```
        input >> _pol[i];
```

```
    }
```

```
    return input;
```

```
}
```





```
ostream& operator << (ostream& output, Polinomio& _pol)
{
    for(int i = 0; i < _pol.n; i++){
        if(i != _pol.n-1) output << _pol[i] << "x^" << i << " + ";
        else output << _pol[i] << "x^" << i << " = 0";
    }
    return output;
}

double& Polinomio::operator[](int pos)
{
    if(pos >= 0 && pos < n) return valores[pos];
    else return valores[0];
}
```


Main



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

using namespace std;

int main()
{
    Polinomio a(3), b(4), c(4);

    cin >> a >> b;
    cout << a << endl;
    cout << b << endl;

    c = b - a;
    cout << c << endl;

    c = a + b;
    cout << c << endl;

    return 0;
}
```

A green document icon with a white label that says 'CPP' in green capital letters.

CPP



```
D:\2021\ecop13\Lab5\Codigos\Ex1\bin\Debug\Ex1.exe
Polinomio: C0 + C1x1 + C2x2 + ... + Cn xn = 0
Digite o valor de C0: 1
Digite o valor de C1: 2
Digite o valor de C2: 3
Digite o valor de C3: 4
Polinomio: C0 + C1x1 + C2x2 + ... + Cn xn = 0
Digite o valor de C0: 3
Digite o valor de C1: 3
Digite o valor de C2: 3
Digite o valor de C3: 3
Digite o valor de C4: 3
1x^0 + 2x^1 + 3x^2 + 4x^3 = 0
3x^0 + 3x^1 + 3x^2 + 3x^3 + 3x^4 = 0
2x^0 + 1x^1 + 0x^2 + -1x^3 + 3x^4 = 0
4x^0 + 5x^1 + 6x^2 + 7x^3 + 3x^4 = 0

Process returned 0 (0x0)   execution time : 30.621 s
Press any key to continue.
```

2ª Questão




Criar uma classe para representar um **ponto cartesiano** no espaço bidimensional (x,y).

Implementar os operadores de pré-incremento e de pós-incremento.

Crie um programa para testar o uso dessa classe com suas funcionalidades.

2ª questão – Exemplo de Solução



```
#ifndef POINT_H
#define POINT_H

#include <iostream>
using namespace std;

class Point {

private:
    int x, y;
public:
    Point(int=0, int=0);
    ~Point() {}
    int get_x() {return x;}
    int get_y() {return y;}

    // primeiro operando é do tipo Point
    Point operator+(Point&);
    Point operator+(int);

    // Primeiro operando é do tipo int
    friend Point operator+(int, Point&);
```



```
// operadores unários
bool operator!() const;

Point& operator++();           //pre-incremento
Point operator++(int);        // pos-incremento

Point& operator--();          //pre-decremento
Point operator--(int);        //pos-decremento

explicit operator int ();     //conversão para int

bool operator==(Point& p);
bool operator!=(Point& p);

friend istream& operator>>(istream& in, Point& p);
friend ostream& operator<<(ostream& out, const Point& p);
};

#endif
```



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

```
Point::Point(int xx, int yy) : x{xx}, y{yy} {}
```

```
// operações do tipo Point + Point
```

```
Point Point::operator+(Point& p)
```

```
{
```

```
    int xx = x + p.x;
```

```
    int yy = y + p.y;
```

```
    return Point{xx, yy};
```

```
}
```

```
// operações do tipo Point + int
```

```
Point Point::operator+(int value)
```

```
{
```

```
    int xx = x + value;
```

```
    int yy = y + value;
```

```
    return Point{xx, yy};
```

```
}
```



```
// Função global: operações do tipo int + Point
```

```
Point operator+(int value, Point& p)
```

```
{  
    int xx = p.x + value;  
    int yy = p.y + value;  
    return Point{xx, yy};  
}
```

```
// operadores unários
```

```
bool Point::operator!() const {  
    if(x == 0 && y == 0) return true;  
    return false;  
}
```

```
Point& Point::operator++() {  
    x++;  
    y++;  
    return *this;  
}
```

```
Point Point::operator++(int value) {  
    Point temp = *this;  
    ++(*this);  
    return temp;  
}
```

```
Point Point::operator--( int value) {  
    Point temp = *this;  
    --(*this);  
    return temp;  
}
```



```
Point& Point::operator--() {
    p.x--;
    p.y--;
    return *this;
}

// função converte Ponto para int
// retornando o valor do módulo da coordenada
Point::operator int()
{
    return sqrt(x*x + y*y);
}

ostream& operator<<(ostream& out, const Point& p) {
    out << "(" << p.x << ", " << p.y << ")";
    return out;
}

istream& operator>>(istream& in, Point& p) {
    in >> p.x >> p.y;
    return in;
}
```




```
bool Point::operator==(Point& p)
{
    if(x == p.x && y == p.y) return true;
    return false;
    // return (x == p.x && y == p.y);
}

bool Point::operator!=(Point& p)
{
    if(!(*this == p)) return true;
    return false;
    // return !(*this == p);
}
```



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

Main



```
int main() {
    Point p1{10,10};
    Point p2{10,10};

    cout << p1 << " e " << p2;
    cout << " sao " << (p1 == p2 ? "iguais" : "diferentes");
    cout << "\n";

    Point p3{10,11};
    if(p1 != p3)
        cout << p1 << " e " << p3 << " sao diferentes.\n";

    cout << "Ponto 1: " << p1 << "\n";

    cout << "Entre com os valores de p2 (x e y): ";
    cin >> p2;
    cout << "Ponto 2: " << p2 << "\n";

    // Para realizar a conversão, utilize o operador
    // de cast, como já estamos acostumados
    cout << "Modulo de " << p2 << ": " << (int)p2 << "\n";

    int x = (int)p2; // necessário tornar explícito
    cout << "Valor de x = " << x << "\n";
}
```

CPP



```
D:\2021\ecop13\Lab5\Codigos\Ex2\bin\Debug\Ex2.exe  —  □  X
(10, 10) e (10, 10) sao iguais
(10, 10) e (10, 11) sao diferentes.
Ponto 1: (10, 10)
Entre com os valores de p2 (x e y): 6 8
Ponto 2: (6, 8)
Modulo de (6, 8): 10
Valor de x = 10

Process returned 0 (0x0)   execution time : 10.121 s
Press any key to continue.
```

3ª Questão



Criar uma classe que represente uma **Data**, sobrecarregar os operadores de entrada e saída (>> e <<), os operadores de incremento na forma pré-fixada e na forma pos-fixada.

Faça um programa que teste as funcionalidades de sua classe.

Obs: Imprimir a data no formato, dia do mês de ano, ex 01 de abril de 2024.

3ª questão – Exemplo de Solução



```
#ifndef Data_H
#define Data_H
```

```
#include <iostream>
```

```
class Data {
    friend std::ostream& operator<<(std::ostream&, const Data&);
public:
    Data(int d = 1, int m = 1, int y = 1900); // default constructor
    void setData(int, int, int);              // set day, month, year
    Data& operator++();                        // prefix increment operator
    Data operator++(int);                      // postfix increment operator
    Data& operator+=(unsigned int);            // adicionar dias, modify object
    static bool anoBissexto(int);              // é ano bissexto?
    bool endOfMonth(int) const;               // é o ultimo dia do mes?
private:
    unsigned int month;
    unsigned int day;
    unsigned int year;

    static const int days[13]; // dias por mes
    void helpIncrement(); // função de apoio para incrementar data
};

#endif
```

```
// * Adaptado de Deitel & Deitel
```

```
*
```



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

// initialize static member; one classwide copy
const int Data::days[13]{
    0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// Data constructor
Data::Data( int day, int month, int year) {
    setData( day, month, year);
}

// set month, day and year
void Data::setData( int dd, int mm, int yy) {
    if (mm >= 1 && mm <= 12)
        month = mm;
    else
        month = 1; // se nao for valido, fixa janeiro
    if (yy >= 1900 && yy <= 2100)
        year = yy;
    else
        year = 2021; // se nao for valido, fixa 2021
    // test para ano bissexto
    if ((month == 2 && anoBissexto(year) && dd >= 1 && dd <= 29) ||
        (dd >= 1 && dd <= days[month]))
        day = dd;
    else
        day = 1; //caso seja inválido fixa o dia 1
}
```





CPP

```
// overloaded prefix increment operator
Data& Data::operator++() {
    helpIncrement(); // increment Data
    return *this; // reference return to create an lvalue
}

// overloaded postfix increment operator; note that the
// dummy integer parameter does not have a parameter name
Data Data::operator++(int) {
    Data temp{*this}; // hold current state of object
    helpIncrement();

    // return unincremented, saved, temporary object
    return temp; // value return; not a reference return
}

// add specified number of days to Data
Data& Data::operator+=(unsigned int additionalDays) {
    for (unsigned int i = 0; i < additionalDays; ++i) {
        helpIncrement();
    }
    return *this; // enables cascading
}

// if the year is a leap year, return true; otherwise, return false
// Se o ano não terminar em 00 e for divisível por 4 dizemos que ele é bissexto.
// Os anos terminados em 00 serão bissextos se a divisão deles por 400 for exata,
// isto é, o resto da divisão precisa ser igual a zero
bool Data::anoBissexto(int testYear) {
    return (testYear % 400 == 0 || (testYear % 100 != 0 && testYear % 4 == 0));
}
```

```

// determine whether the day is the last day of the month
bool Data::endOfMonth(int testDay) const {
    if (month == 2 && anoBissesto(year)) {
        return testDay == 29; // last day of Feb. in leap year
    }
    else {
        return testDay == days[month];
    }
}

```

```

// function to help increment the Data
void Data::helpIncrement() {
    // day is not end of month
    if (!endOfMonth(day)) {
        ++day; // increment day
    }
    else {
        if (month < 12) { // day is end of month and month < 12
            ++month; // increment month
            day = 1; // first day of new month
        }
        else { // last day of year
            ++year; // increment year
            month = 1; // first month of new year
            day = 1; // first day of new month
        }
    }
}

```





```
// overloaded output operator
ostream& operator<<(ostream& output, const Data& d) {
    static string monthName[13]{ "", "Janeiro", "Fevereiro",
        "Março", "Abril", "Maio", "Junho", "Julho", "Agosto",
        "Setembro", "Outubro", "Novembro", "Dezembro" };
    output << d.day << " de " << monthName[d.month] << " de " << d.year;
    return output; // enables cascading
}

// * Adaptado de Deitel & Deitel
```

```
#include <iostream>
#include "Data.h" // Date class definition
using namespace std;
```

Main



```
int main() {
    Data d1{27, 12, 2010}; // December 27, 2010
    Data d2; // defaults 1/01/1900

    cout << "d1 is " << d1 << "\nd2 is " << d2;
    cout << "\n\nd1 += 7 is " << (d1 += 7);

    d2.setData(28, 2, 2008);
    cout << "\n\n d2 is " << d2;
    cout << "\n++d2 is " << ++d2 << " (ano bissexto permite dia 29)";

    Data d3{13, 7, 2010};

    cout << "\n\nTeste da forma pre-fixada do operador de incremento :\n"
        << " d3 is " << d3 << endl;
    cout << "++d3 is " << ++d3 << endl;
    cout << " d3 is " << d3;

    cout << "\n\nTeste da forma pos-fixada do operador de incremento: \n"
        << " d3 is " << d3 << endl;
    cout << "d3++ is " << d3++ << endl;
    cout << " d3 is " << d3 << endl;
}
```

```
// * Adaptado de Deitel & Deitel
```

CPP



```
D:\2021\ecop13\Lab5\Codigos\Ex3\bin\Debug\Ex3.exe
d1 is 27 de Dezembro de 2010
d2 is 1 de Janeiro de 1900

d1 += 7 is 3 de Janeiro de 2011

    d2 is 28 de Fevereiro de 2008
++d2 is 29 de Fevereiro de 2008 (ano bissexto permite dia 29)

Teste da forma pre-fixada do operador de incremento :
    d3 is 13 de Julho de 2010
++d3 is 14 de Julho de 2010
    d3 is 14 de Julho de 2010

Teste da forma pos-fixada do operador de incremento:
    d3 is 14 de Julho de 2010
d3++ is 14 de Julho de 2010
    d3 is 15 de Julho de 2010

Process returned 0 (0x0)    execution time : 0.064 s
Press any key to continue.
```

4ª Questão



Sobrecarregar os operadores de conversão de tipo para a classe fração e para a classe complexo, dos laboratórios anteriores.

Permitindo também que uma fração possa ser transformada em complexo.

5ª Questão



Declarar e implementar uma classe que represente um **livro** para um sistema a ser implementado para uma biblioteca.

Para pensar: Que alterações seriam necessárias nessa classe para que ela representasse um livro para um sistema implementado em uma livraria.