ECOP13A-Lab7

Guia de Laboratório

Prof. André Bernardi
andrebernardi@unifei.edu.br

7º Laboratório ECOP13A
30 de maio 2025

1ª Questão

Construir as classes para representar uma hierarquia **Politico** / **Presidente** / **Governador** / **Prefeito**.

- Acrescente uma função Imprime() em cada uma das classes.
- No construtor de cada classe, acrescente mensagens de depuração para saber por onde o programa está passando enquanto é executado.
- Utilize as funções definidas nas classes bases dentro das classes derivadas.

1ª questão
Exemplo de Solução


```
#ifndef POLITICO_H
#define POLITICO_H

#include <string>
#include <iostream>

using namespace std;

class Politico{
    protected:
        string nome, partido, numero;
    public:
        Politico(string n, string p, string nu): nome{n}, partido{p}, numero{nu} {
                cout << " Construindo Politico!" << endl;
        }
     void imprime();
          ~Politico( ) { cout << "Destroi Politico!" << endl << endl; }
};

class Presidente: public Politico{
    protected:
        string pais;
    public:
        Presidente(string nome, string partido, string numero, string p): Politico{nome, partido,
numero},  pais{p} {
            cout << " Construindo Presidente!" << endl;
        }
        void imprime();
        ~Presidente() { cout << "Destroi Presidente!" << endl; }
};
```

```
class Governador: public Presidente{
    protected:
        string estado;
    public:
        Governador(string nome, string partido, string numero,
                   string pais, string e):
                   Presidente{nome, partido, numero, pais},
                   estado{e} {
                cout << "Construindo Governador!" << endl;
         }
         void imprime();
         ~Governador() { cout << "Destroi Governador!" << endl; }
};

class Prefeito: public Governador{
    protected:
         string cidade;
    public:
         Prefeito(string nome, string partido, string numero,
                   string pais, string estado, string c):
                   Governador{nome, partido, numero, pais, estado},
                   cidade{c} {
              cout << " Construindo Prefeito!" << endl << endl;
         }
         void imprime();
         ~Prefeito() { cout << "Destroi Prefeito!" << endl; }
};
#endif // POLITICO_H
```

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

using namespace std;

void Politico::imprime(){

    cout << "Numero:      "    << numero << endl;
    cout << "Nome:        "    << nome << endl;
    cout << "Partido:     "    << partido << endl;
}

void Presidente::imprime(){
    Politico::imprime();
    cout << "Pais:        "    << pais << endl;
}

void Governador::imprime(){
    Presidente::imprime();
    cout << "Estado:      "   << estado << endl;
}

void Prefeito::imprime(){
    Governador::imprime();
    cout << "Cidade:      "   << cidade << endl << endl;
}
```

Main


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

int main(){
    cout << "## POLITICO COMPLETO ##" << endl << endl;
    Prefeito p("Nome1", "P1", "1", "BRASIL", "MG", "Itajuba");

    cout << "********************" << endl << endl;
    cout << "# Prefeito 01 #" << endl;
    p.imprime();

    Governador g("Nome2", "P2", "2", "Brasil", "QQuer");
    cout << endl << "# Governador 01 #" << endl;
    g.imprime(); cout << endl;

    Presidente pr("Nome3",  "P3", "3", "Brasil");
    cout << endl << "# PRESIDENTE 01 #" << endl;
    pr.imprime(); cout << endl;

    Politico pi("Nome", "P4", "0");
    cout << endl << "# POLITICO 01 #" << endl;
    pi.imprime(); cout << endl;
    cout << "********************" << endl << endl;

    return 0;
}
```

2ª Questão

Crie uma hierarquia de classes para representar a hierarquia **Ponto/Circulo/Cilindro**.

Considere que o Cilindro é um Circulo com altura diferente de zero e que o Circulo é um Ponto com raio diferente de zero.

Além dos construtores, métodos de acesso, operadores de leitura (>>) e impressão (<<), implemente as funções *area*() e *volume*() para a hierarquia.

2ª questão – Exemplo de Solução
Classe Ponto

```
#include <iostream>
using namespace std;
#define PI 3.1415

#ifndef CIRCULO_H
#define CIRCULO_H

class Ponto
{
protected:
    double x, y;
public:
    Ponto(double x = 0, double y = 0): x{x}, y{y} { }
    ~Ponto() {}

     void read()          { cin >> x >> y; }
     void print() const   { cout << "C(" << x << "," << y << ")"; }

     double area()    { return 0; }
     double volume()  { return 0; }

     friend istream& operator>>(istream& input, Ponto& in);
         friend ostream& operator<<(ostream& output, const Ponto& out);
};
```

2ª questão – Exemplo de Solução
Classe Circulo


```
class Circulo: public Ponto
{
protected:
        double raio;
public:
   Circulo(double x = 0, double y = 0, double r = 0): Ponto{x, y}, raio {r} {}
   ~Circulo() {}

   void read()         { cin >> x >> y >> raio; }
   void print() const { Ponto::print();  cout << " RAIO = " << raio; }

   double area()    { return PI*raio*raio; }
   double volume() { return 0; }

};
```

2ª questão – Exemplo de Solução
Classe Cilindro


```
class Cilindro: public Circulo
{
protected:
        double altura;
public:
   Cilindro(double x = 0, double y = 0, double raio = 0, double a = 0):
                    Circulo{x, y, raio}, altura{a} {}
   ~Cilindro() {}

   void read()         { cin >> x >> y >> raio >> altura; }
   void print() const { Circulo::print();  cout << " ALTURA = " << altura; }

   double area()    { return (2*Circulo::area())+ 2*PI*raio*altura ; }
   double volume() { return (Circulo::area())*altura; }
};

#endif // CIRCULO_H
```


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

using namespace std;


istream& operator>>(istream& input, Ponto& pt)
{
   pt.read();
   return input;
}

ostream& operator<<(ostream& output, const Ponto& pt)
{
   pt.print();
   return output;
}
```

Main


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

using namespace std;

int main(){

    //Cilindro é um Circulo com altura diferente de zero e
    //que o Circulo é um Ponto de raio diferente de zero

    Ponto r(1, 2);
    cout << "Ponto = " << r << endl;

    Circulo c(3, 4, 5);
    cout << "Circulo = " << c << endl;

    Cilindro cl(7, 8, 9, 10);
    cout << "Cilindro = " << cl << endl;

    cout << endl;
    cout << "Area circulo = "    << c.area() << endl;
    cout << "Volume cilindro = " << cl.area() << endl;

    Ponto p;
    cout << endl << "Digite x e y para o centro do ponto: ";
    cin >> p;
    cout << "Digitado foi " << p << endl;
```

Main


```
   Circulo circ;
   cout << endl << "Digite x e y do centro e o raio: ";
   cin >> circ;
   cout << "Digitado foi " << circ << endl;

   Cilindro cilin;
   cout << endl << "Digite x e y do centro, raio e altura: ";
   cin >> cilin;

   cout << "Digitado foi " << cilin << endl;

   return 0;
}
```

3ª Questão

Utilizar a classe polinômio do laboratório 5 para implementar uma função que encontre pelo menos uma raiz real dele, se ela existir utilizando o método de Newton:

Para encontrar uma raiz real de um polinômio: (n > 2),

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

pode-se aplicar o método de Newton, que consiste em refinar uma aproximação inicial x_0 dessa raiz através da expressão:

$$x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)}$$

onde:

n = 0,1,2,...,

p'(x) é a primeira derivada de p(x).

Usualmente, repete-se esse refinamento até que $|x_{n+1} - x_n| < \varepsilon$, para $\varepsilon > 0$, ou até que *m* iterações tenham sido executadas.

Implemente na classe Polinômio as seguintes funções:

- Dado um polinômio p(x), calcule e retorne a sua derivada p'(x).
- Dado um polinômio p(x), calcule seu valor em um ponto. Utilize essa função para calcular $p(x_n)$ e $p'(x_n)$ em cada iteração.
- Dado um polinômio p(x), uma aproximação inicial x_0 e o número máximo *m* de iterações que devem ser executadas, calcule uma raiz real pelo método de Newton, se ela existir.

Exemplo numérico

- Considere o polinômio $p(x)= x^2 - 5x + 6$, que possui duas raízes reais em 2 e 3. Sua derivada é $p'(x)=2x-5$.

Para testar o método de Newton podemos supor uma aproximação inicial $x_0 = 500$ e observar sua convergência.

$x_1= 500 – p(500)/p'(500) = 500 – 247506/995 =$ **251.250251256**

$x_2= x_1 - p(x_1) / p'(x_1) = 251.250251256 –$ $61876.4374999/497.5005025=$ **126.87**

$x_3= x_2 - p(x_2) / p'(x_2) = 126.87562814 – 16097.42/248.75 =$ **62.1626885688**

...

$x_n= 3$

3ª questão
Exemplo de Solução


```
#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&);

    Polinomio derivada( );
    double raiz( int aprox_inicial, int num_iteracoes );
    double calcula( double x );
};
#endif
```

```cpp
#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 _n)
{
        n = _n + 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
   }
}

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[i] = _pol.valores[i];
   else
       for(int j = i; j < n; j++)
           temp[i] = valores[i];
   return temp;
}
```

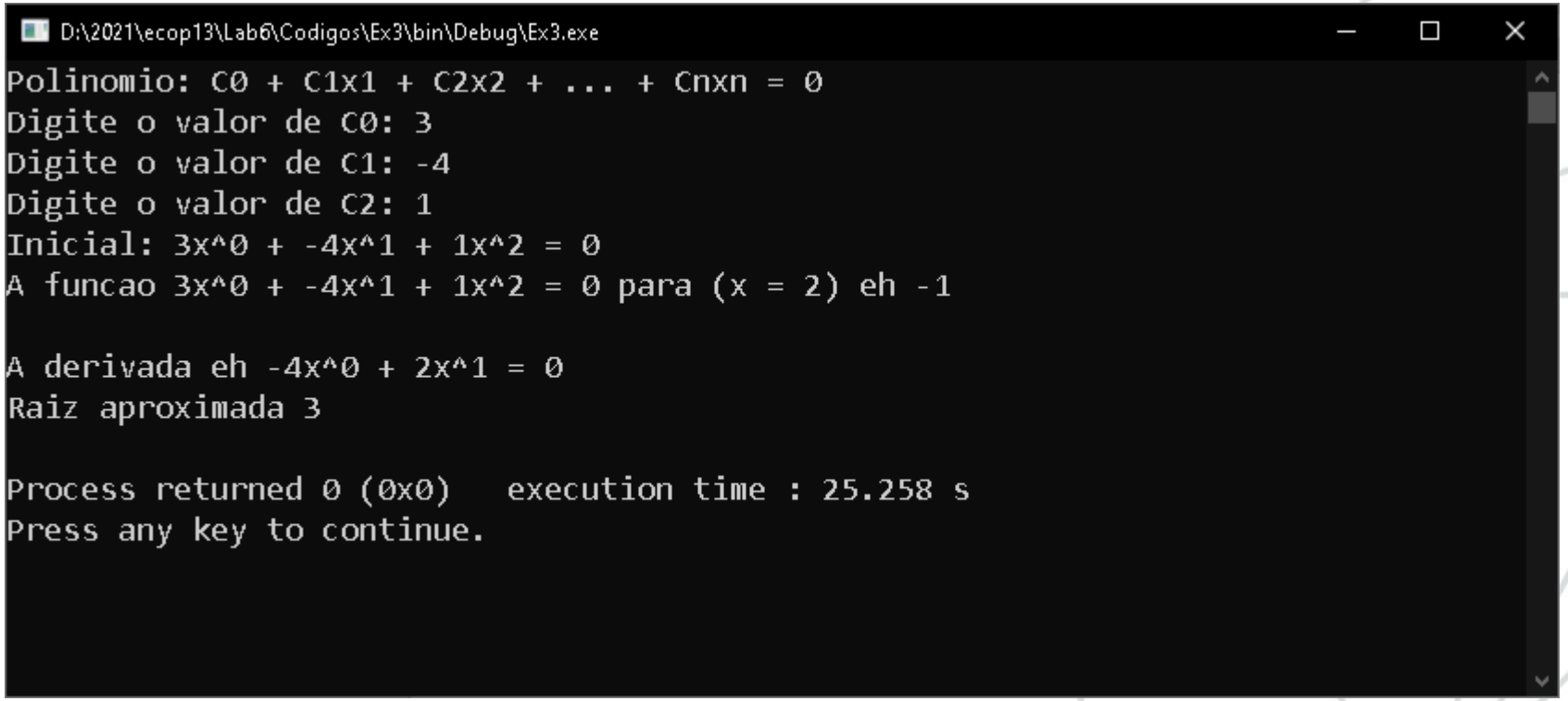
```
Polinomio Polinomio::operator-(Polinomio _pol)
{
      Polinomio temp(max(_pol.n, n)-1);

      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[i] = - _pol.valores[i];
      else
         for(int j = i; j < n; j++)
            temp[i] = valores[i];
      return temp;
}

istream& operator >> (istream& input, Polinomio& _pol)
{
      cout << "Polinomio: C0 + C1x^1 + C2x^2 + ... + Cnx^n = 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];//alterar para lançar exceção
}
```

```
Polinomio Polinomio::derivada(){
    Polinomio tmp(n-1-1);
    for(int i = 1; i < n ; i++){
        tmp.valores[i-1] = i * valores[i];
    }
    return tmp;
}
double Polinomio::raiz( int n_inicial, int num_iteracoes ){

    Polinomio inicio =  (*this) ; // cria uma cópia local
    Polinomio derivada_inicio; // cria polinomio para ser a derivada
    derivada_inicio = inicio.derivada();

    double root;
    for(int i = 0; i < num_iteracoes; i++ ){
        root = n_inicial - inicio.calcula( n_inicial) /
                                derivada_inicio.calcula( n_inicial );
        n_inicial = root;
    }
    return root;
}
double Polinomio::calcula(double x){
    double resp = 0;
    for(int i = 0; i < n ; i++){
        resp += valores[i] * pow(x, i);
    }
    return resp;
}
```

Main

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

using namespace std;

int main()
{
 //Cria polinomio grau 2
    Polinomio p0(2);
    cin >> p0;

    cout << "Inicial: " << p0 << endl;
    cout << "A funcao " <<  p0 << " para (x = 2) eh " << p0.calcula( 2 ) << endl;

    // mostra derivada
    Polinomio derivada = p0.derivada();
    cout << endl << "A derivada eh " << derivada << endl;

    // valores para o metodo de Newton
    int n_inicial = 20, iteracoes = 2000000;
    cout << "Raiz aproximada " << p0.raiz(n_inicial, iteracoes) << endl;
}
```


D:\2021\ecop13\Lab6\Codigos\Ex3\bin\Debug\Ex3.exe
Polinomio: C0 + C1x1 + C2x2 + ... + Cnxn = 0
Digite o valor de C0: 3
Digite o valor de C1: -4
Digite o valor de C2: 1
Inicial: 3x^0 + -4x^1 + 1x^2 = 0
A funcao 3x^0 + -4x^1 + 1x^2 = 0 para (x = 2) eh -1
A derivada eh -4x^0 + 2x^1 = 0
Raiz aproximada 3
Process returned 0 (0x0)   execution time : 25.258 s
Press any key to continue.

4ª Questão

Analisar a classe de exemplo do Livro do Deitel que representa um número de telefone formatado, e alterar essa classe para que funcione com o formato utilizado no Brasil.


```
// Fig. 11.3: PhoneNumber.h
// PhoneNumber class definition
#ifndef PHONENUMBER_H
#define PHONENUMBER_H

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

class PhoneNumber
{
   friend ostream &operator<<( ostream &, const PhoneNumber & );
   friend istream &operator>>( istream &, PhoneNumber & );
private:
   string areaCode;  // 3-digit area code
   string exchange;  // 3-digit exchange
   string line;      // 4-digit line
}; // end class PhoneNumber

#endif
```

```
// Fig. 11.4: PhoneNumber.cpp
// Overloaded stream insertion and stream extraction operators
// for class PhoneNumber.
#include <iomanip>
#include "PhoneNumber.h"
using namespace std;

// overloaded stream insertion operator; cannot be
// a member function if we would like to invoke it with
// cout << somePhoneNumber;
ostream &operator<<( ostream &output, const PhoneNumber &number ){
   output << "(" << number.areaCode << ") "
      << number.exchange << "-" << number.line;
   return output; // enables cout << a << b << c;
} // end function operator<<

// overloaded stream extraction operator; cannot be
// a member function if we would like to invoke it with
// cin >> somePhoneNumber;
istream &operator>>( istream &input, PhoneNumber &number ){
   input.ignore();                          // skip (
   input >> setw( 3 ) >> number.areaCode;   // input area code
   input.ignore( 2 );                       // skip ) and space
   input >> setw( 3 ) >> number.exchange;   // input exchange
   input.ignore();                          // skip dash (-)
   input >> setw( 4 ) >> number.line;       // input line
   return input;                            // enables cin >> a >> b >> c;
} // end function operator>>
```

```cpp
// Fig. 11.5: fig11_05.cpp
// Demonstrating class PhoneNumber's overloaded stream insertion
// and stream extraction operators.
#include <iostream>
#include "PhoneNumber.h"
using namespace std;

int main()
{
   PhoneNumber phone; // create object phone

   cout << "Enter phone number in the form (123) 456-7890:" << endl;

   // cin >> phone invokes operator>> by implicitly issuing
   // the global function call operator>>( cin, phone )
   cin >> phone;

   cout << "The phone number entered was: ";

   // cout << phone invokes operator<< by implicitly issuing
   // the global function call operator<<( cout, phone )
   cout << phone << endl;
} // end main
```