Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

`IE-0217` Estructuras abstractas de datos y algoritmos para ingeniería.

Laboratorio 1: Programación Genérica

Belinda Brown Ramírez - `B61254`

I-2019

Tabla de contenidos

1. Enunciado

Diseñe, desarrolle y documente una serie de clases con plantillas que permitan hacer operaciones básicas con enteros, reales, fracciones y polinomios.

1.1. Clase emplantillada

Creen una clase base llamada Calculadora.Tanto la clase como sus métodos deben ser emplatillados para un typename Data. En esta clase crear cinco métodos llamados:

- add(Data &di, const Data &d2)
- sub(Data &di, const Data &d2)
- mul(Data &di, const Data &d2)
- div(Data &di, const Data &d2)
- print(Data &d)

Tanto el constructor como el destructor son vacíos.

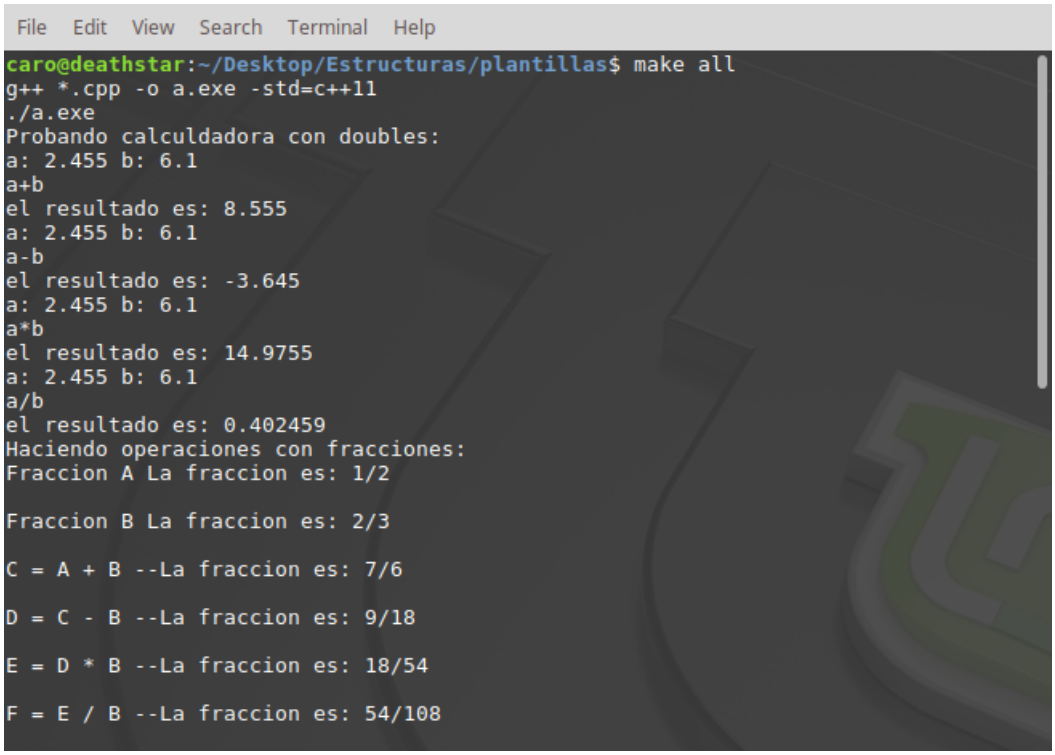
1.2. Clases operadoras

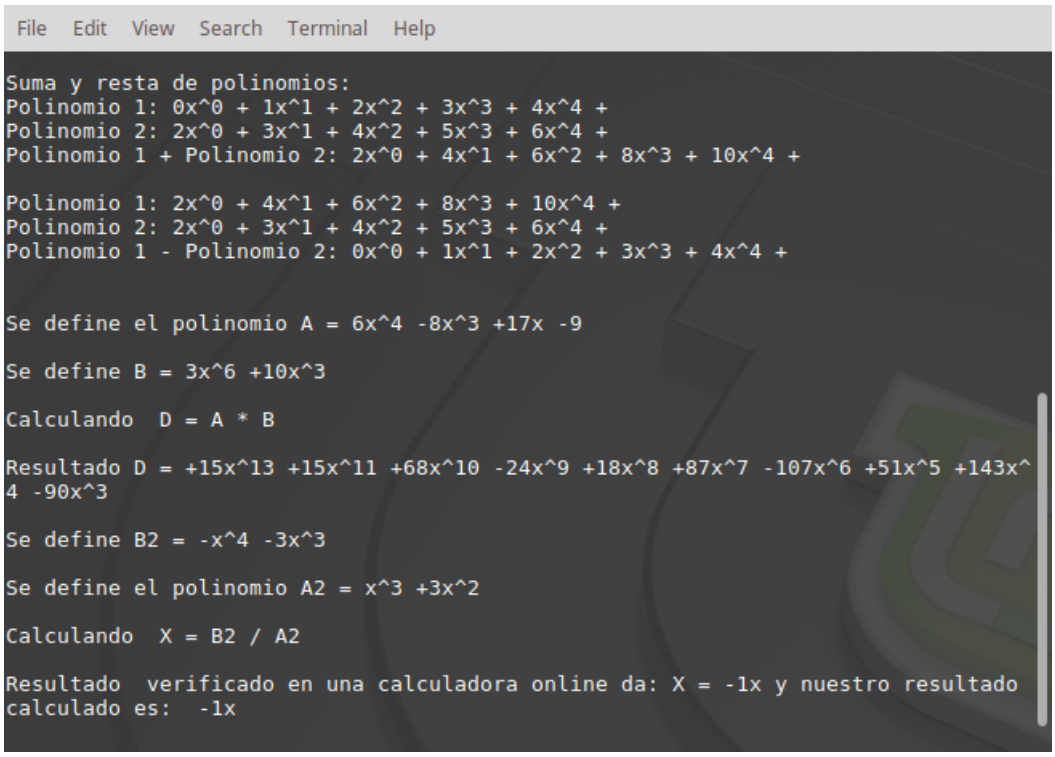
Construya una clase Polinomio y una clase Fraccion. Para cada una de estas clases, sobrecargue los siguiente operadores:

- void operator+(const CLASE &rhs)
- void operator-(const CLASE &rhs)
- void operator*(const CLASE &rhs)
- void operator(const CLASE &rhs)
- CLASE& operator=(const CLASE &rhs)
- string operator (para impresión)

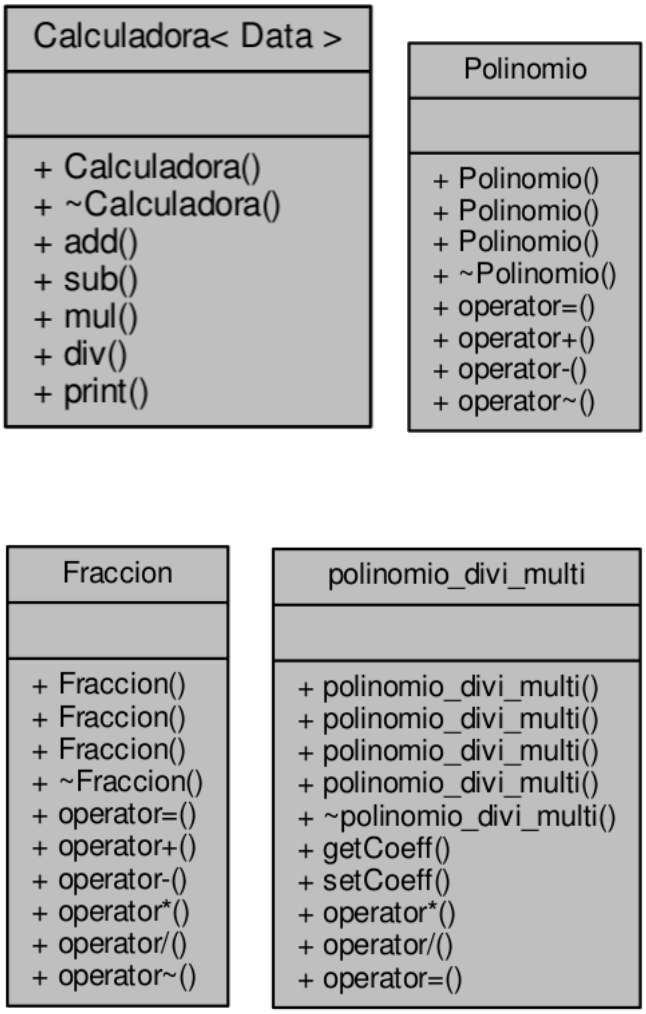
Las operaciones son autoexplicativas.

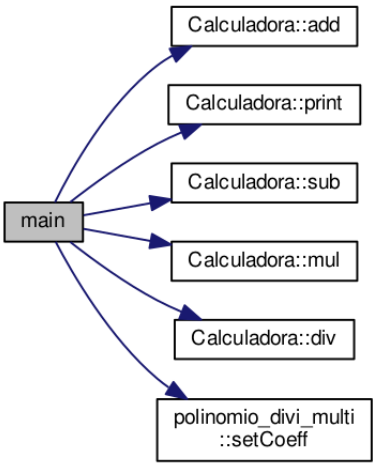
2. Pruebas:

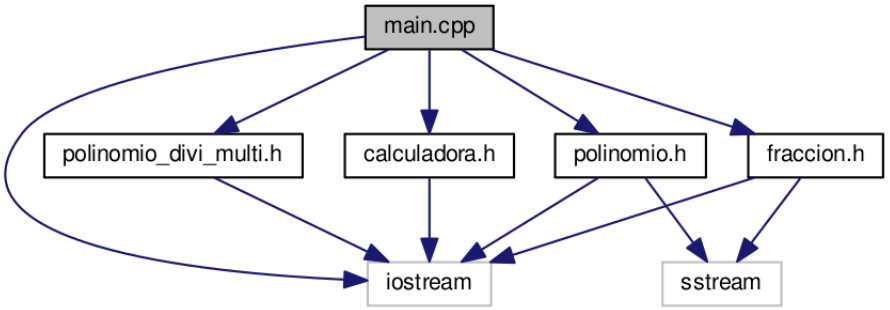

File Edit View Search Terminal Help
caro@deathstar:~/Desktop/Estructuras/plantillas$ make all
g++ *.cpp -o a.exe -std=c++11
./a.exe
Probando calculdadora con doubles:
a: 2.455 b: 6.1
a+b
el resultado es: 8.555
a: 2.455 b: 6.1
a-b
el resultado es: -3.645
a: 2.455 b: 6.1
a*b
el resultado es: 14.9755
a: 2.455 b: 6.1
a/b
el resultado es: 0.402459
Haciendo operaciones con fracciones:
Fraccion A La fraccion es: 1/2
Fraccion B La fraccion es: 2/3
C = A + B --La fraccion es: 7/6
D = C - B --La fraccion es: 9/18
E = D * B --La fraccion es: 18/54
F = E / B --La fraccion es: 54/108


File Edit View Search Terminal Help
Suma y resta de polinomios:
Polinomio 1: 0x^0 + 1x^1 + 2x^2 + 3x^3 + 4x^4 +
Polinomio 2: 2x^0 + 3x^1 + 4x^2 + 5x^3 + 6x^4 +
Polinomio 1 + Polinomio 2: 2x^0 + 4x^1 + 6x^2 + 8x^3 + 10x^4 +
Polinomio 1: 2x^0 + 4x^1 + 6x^2 + 8x^3 + 10x^4 +
Polinomio 2: 2x^0 + 3x^1 + 4x^2 + 5x^3 + 6x^4 +
Polinomio 1 - Polinomio 2: 0x^0 + 1x^1 + 2x^2 + 3x^3 + 4x^4 +
Se define el polinomio A = 6x^4 -8x^3 +17x -9
Se define B = 3x^6 +10x^3
Calculando D = A * B
Resultado D = +15x^13 +15x^11 +68x^10 -24x^9 +18x^8 +87x^7 -107x^6 +51x^5 +143x^
4 -90x^3
Se define B2 = -x^4 -3x^3
Se define el polinomio A2 = x^3 +3x^2
Calculando X = B2 / A2
Resultado verificado en una calculadora online da: X = -1x y nuestro resultado
calculado es: -1x

3. Diagramas de clase







4. Anexos

4.1. Código Calculadora:

calculadora.h:

```
#ifndef CALCULADORA_H
#define CALCULADORA_H

#include <iostream>

using namespace std;

template <typename Data>
class Calculadora {
    public:
        Calculadora(){};
        ~Calculadora(){};
        Data add(Data &d1, const Data &d2){return d1+d2;}
        Data sub(Data &d1, const Data &d2){return d1-d2; }
        Data mul(Data &d1, const Data &d2){return d1*d2;}
        Data div(Data &d1, const Data &d2){return d1/d2;}
        void print(Data &d1){
            cout << "el resultado es: " << d1 <<endl;
        }
};

#endif
```

4.2. Código Polinomio:

polinomio.h:

```
#ifndef POLINOMIO_H
#define POLINOMIO_H

#include <iostream>
```

```
#include <sstream>

using namespace std;

class Polinomio {
    private:
        int s;
        int* g;

    public:
        Polinomio();
        Polinomio(int s, int* g);
        Polinomio(const Polinomio &orig);
        ~Polinomio();
        Polinomio& operator=(const Polinomio &rhs);
        void operator+(const Polinomio &rhs);
        void operator-(const Polinomio &rhs);
        string operator~();
    };

#endif
```

polinomio.cpp:

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

Polinomio::Polinomio() {
};
Polinomio::Polinomio(int s, int* g) {
    this->s=s;
    this->g=g;
};
Polinomio::Polinomio(const Polinomio &orig) {
    this->s = orig.s;
    for(int i =0; i<s; i++){
        this->g[i]=orig.g[i];
    }

};
Polinomio::~Polinomio() {
};

Polinomio& Polinomio::operator=(const Polinomio &rhs) {
    this->s = rhs.s;
    this->g = new int[s];
    for(int i=0; i<s; i++){
        this->g[i]=rhs.g[i];
    }
    return *this;
};

void Polinomio::operator+(const Polinomio &rhs){
    if (s>=rhs.s){
        for(int i =0; i<s; i++){
            this->g[i]=this->g[i] + rhs.g[i];
        }
    }
```

```
    else{
        for(int i=0; i<rhs.s ; i++ ){
            rhs.g[i]=this->g[i] + rhs.g[i];
            delete this->g;
            this->g=new int[rhs.s];
            for (int i=0; i<rhs.s ; i++ ){
                this->g[i]=rhs.g[i];
            }
        }
    }
};
void Polinomio::operator-(const Polinomio &rhs){
    if (s>=rhs.s){
        for(int i =0; i<s; i++){
            this->g[i]=this->g[i] - (rhs.g[i]);
        }
    }
    else{
        for(int i=0; i<rhs.s ; i++ ){
            rhs.g[i]= rhs.g[i] - (this->g[i]);
            }
        delete this->g;
        this->g=new int[rhs.s];
        for (int i=0; i<rhs.s ; i++ ){
            this->g[i]=rhs.g[i];
            }
        }
        };

string Polinomio::operator~(){
    stringstream s("", ios_base::app | ios_base::out);
    int k =(this->s)-1;
    for(int i =0; i<(this->s);i++){
        if (i<=(k)){
            s << to_string(g[i]) << "x^" <<i << " + ";
        }
        else if (i==k){
            s << to_string(g[i]) << "x^" <<i;
        }
        }
        s <<endl;
        return s.str();
        };
```

```

#ifndef POLINOMIO_DIVI_MULTI_H
#define POLINOMIO_DIVI_MULTI_H


#include <iostream>


class polinomio_divi_multi
{
    //Sobrecarga de operador
    friend std::istream& operator>>(std::istream &inStream, polinomio_divi_multi &
    sourceCoeff);

    // Imprime el polinomio
```

```
15      friend std::ostream& operator<<(std::ostream &outStream, const
        polinomio_divi_multi &Sourcepolinomio_divi_multi);
16
17  public:
18      // contructores y destructores
19      polinomio_divi_multi();
20      polinomio_divi_multi(int coeff);
21      polinomio_divi_multi(int coeff, int power);
22      polinomio_divi_multi(const polinomio_divi_multi &source);
23      ~polinomio_divi_multi();
24
25      // Obtiene y tiene
26      int getCoeff(int power) const;
27      void setCoeff(int coeff, int power);
28
29      polinomio_divi_multi operator*(const polinomio_divi_multi &rhs) const;
30      polinomio_divi_multi operator/(const polinomio_divi_multi &rhs) const;
31      polinomio_divi_multi& operator=(const polinomio_divi_multi &rhs);
32
33  private:
34      int* coeffPtr;  // Puntero al arreglo
35      int size;   //Tama o del arreglo
36  };
37
38
39  #endif
```

```
1
2   #include "polinomio_divi_multi.h"
3
4   using namespace std;
5
6   istream &operator>>(istream &inStream, polinomio_divi_multi &sourceCoeff)
7   {
8       int coeff;
9       int power;
10
11      while (true)
12      {
13          inStream >> coeff >> power;
14
15          if ((coeff == -1) && (power == -1))
16          {
17              break;
18          }
19
20          sourceCoeff.setCoeff(coeff, power);
21      }
22
23      return inStream;
24  }
25
26  ostream &operator<<(ostream &outStream, const polinomio_divi_multi &
        sourcepolinomio_divi_multi)
27  {
28      bool allZeroes = true;
29
30          for (int i = sourcepolinomio_divi_multi.size - 1; i >= 0; i--)
31
```

```
32          {
33              if (sourcepolinomio_divi_multi.coeffPtr[i] != 0)
34              {
35                  allZeroes = false;
36
37                  outStream << " ";
38
39                  if (sourcepolinomio_divi_multi.coeffPtr[i] > 0)
40                  {
41                      outStream << "+";
42                  }
43
44                  outStream << sourcepolinomio_divi_multi.coeffPtr[i];
45
46                  if (i != 0 && i != 1)
47                  {
48                      outStream << "x^" << i;
49                  }
50
51                  if (i == 1)
52                  {
53                      outStream << "x";
54                  }
55              }
56          }
57
58      if (allZeroes == false)
59      {
60          return outStream;
61      }
62      else
63      {
64          return outStream << "0";
65      }
66  }
67
68
69  polinomio_divi_multi::polinomio_divi_multi()
70  {
71      this->size = 1;
72      coeffPtr = new int[this->size];
73      coeffPtr[0] = 0;
74  }
75
76
77  polinomio_divi_multi::polinomio_divi_multi(int coeff)
78  {
79      this->size = 1;
80      coeffPtr = new int[this->size];
81      coeffPtr[0] = coeff;
82  }
83
84
85  polinomio_divi_multi::polinomio_divi_multi(int coeff, int power)
86  {
87      this->size = power + 1;
88      coeffPtr = new int[this->size];
89
90      for (int i = 0; i < this->size ; i++)
```

```
91      {
92          coeffPtr[i] = 0;
93      }
94
95      coeffPtr[power] = coeff;
96  }
97  polinomio_divi_multi::polinomio_divi_multi(const polinomio_divi_multi &source)
98  {
99      this->size = source.size;
100     coeffPtr = new int[size];
101
102     for (int i = 0; i < this->size; i++)
103     {
104         coeffPtr[i] = source.coeffPtr[i];
105     }
106 }
107
108
109 polinomio_divi_multi::~polinomio_divi_multi()
110 {
111     delete[] coeffPtr;
112     coeffPtr = NULL;
113 }
114
115
116 int polinomio_divi_multi::getCoeff(int power) const
117 {
118     if ((power >= 0) && (power < this->size))
119     {
120         return coeffPtr[power];
121     }
122     else
123     {
124         return 0;
125     }
126 }
127
128 void polinomio_divi_multi::setCoeff(int coeff, int power)
129 {
130     if (power >= 0)
131     {
132         if (power < this->size)
133         {
134             coeffPtr[power] = coeff;
135         }
136         else
137         {
138             int *tempArr = new int[power + 1];
139
140             for (int i = 0; i < this->size; i++)
141             {
142                 tempArr[i] = coeffPtr[i];
143             }
144
145             for (int j = this->size; j < power + 1; j++)
146             {
147                 tempArr[j] = 0;
148             }
149
```

```
150             tempArr[power] = coeff;
151 
152             delete[] coeffPtr;
153             coeffPtr = NULL;
154             coeffPtr = tempArr;
155             tempArr = NULL;
156             this->size = power + 1;   // redisenando la entrega
157         }
158     }
159 }
160 
161 polinomio_divi_multi polinomio_divi_multi::operator*(const polinomio_divi_multi &rhs
    ) const
162 {
163     int tempSize;
164     tempSize = (this->size + rhs.size - 2);
165 
166     polinomio_divi_multi tempArr(0, tempSize);
167 
168         for (int i = 0; i < this->size; i++)
169         {
170             if (this->coeffPtr[i] != 0) // solo multiplica sino es por cero
171             {
172                 for (int j = 0; j < rhs.size; j++)
173                 {
174                     tempArr.coeffPtr[i + j] += (this->coeffPtr[i] * rhs.coeffPtr[j])
    ;
175                     //
176                 }
177             }
178         }
179 
180     return tempArr; // dveuleve el producro de la multiplicacion de dos
    polinomio_divi_multinomials
181 }
182 
183 
184 polinomio_divi_multi polinomio_divi_multi::operator/(const polinomio_divi_multi &rhs
    ) const
185 {
186     int tempSize;
187     tempSize = (this->size + rhs.size);
188 
189     polinomio_divi_multi tempArr(0, tempSize);
190 
191         for (int i = 0; i < this->size; i++)
192         {
193             if (this->coeffPtr[i] != 0) //solo divide sino es por cero
194             {
195                 for (int j = 0; j < rhs.size; j++)
196                 {
197                     tempArr.coeffPtr[j-i-2] += (rhs.coeffPtr[j] /  this->coeffPtr[i
    ]);
198                     // rellena los nuevos valores
199                 }
200             }
201         }
202 
203     return tempArr; // retorna el producto de la divide de dos polinomio_divi_multi
```

```
}
```

```
polinomio_divi_multi& polinomio_divi_multi::operator=(const polinomio_divi_multi &
    rhs)
{
    if (this->coeffPtr == rhs.coeffPtr)
    {
        return *this;
    }

    if (this->size < rhs.size)
    {
        delete[] this->coeffPtr;    // elimina este arreglo
        coeffPtr = NULL;
        this->size = rhs.size;
        this->coeffPtr = new int[this->size];   // crea una copia para el nuevo
    arreglo

        for (int i = 0; i < this->size; i++)
        {
            this->coeffPtr[i] = rhs.coeffPtr[i];    // copia los valores en el nuevo
     arreglo
        }
    }

    if (this->size > rhs.size)  // se fija si el arreglo es mas grande
    {
        for (int i = 0; i < this->size; i++)
        {
            this->coeffPtr[i] = 0;  // pone todos los indices en cero
        }

        for (int j = 0; j < rhs.size; j++)
        {
            this->coeffPtr[j] = rhs.coeffPtr[j]; // copia los valores en el nuevo
    arreglo
        }
    }

    if (this->size == rhs.size) // se fija si el largo del arreglo es el mismo
    {
        for (int k = 0; k < this->size; k++)
        {
            this->coeffPtr[k] = rhs.coeffPtr[k];    //copia valores para el nuevo
    arreglo
        }
    }

    return *this;   // devuelve el nuevo arreglo
}
```

4.3. Código Fracciones:

fraccion.h:

```
#ifndef FRACCION_H
#define FRACCION_H

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


class Fraccion {
    private:
        int n;
        int d;

    public:
        Fraccion() ;
        Fraccion(int n, int d) ;
        Fraccion(const Fraccion &orig) ;
        ~Fraccion();
        Fraccion& operator=(const Fraccion &rhs) ;
        void operator+(const Fraccion &rhs);
        void operator-(const Fraccion &rhs);
        void operator*(const Fraccion &rhs);
        void operator/(const Fraccion &rhs);
        string operator~();
    };

#endif
```

fraccion.cpp:

```
#include <iostream>
#include <sstream>
#include "fraccion.h"

using namespace std;


Fraccion::Fraccion() {};

Fraccion::Fraccion(int n, int d) {
    this->n = n; //Numerador
    this->d=d;   //Denominador
};
Fraccion::Fraccion(const Fraccion &orig) {
    this->n = orig.n;
    this->d = orig.d;

};
Fraccion::~Fraccion() {
};
Fraccion& Fraccion::operator=(const Fraccion &rhs) {
    this->n = rhs.n;
    this->d = rhs.d;
    return *this;
};
```

```
void Fraccion::operator+(const Fraccion &rhs){
    this->n = (this->n * rhs.d) + (this->d * rhs.n) ;
    this->d = this->d * rhs.d;
};

void Fraccion::operator-(const Fraccion &rhs){
    this->n = (this->n * rhs.d) - (this->d * rhs.n) ;
    this->d = this->d * rhs.d;
};
void Fraccion::operator*(const Fraccion &rhs){
    this->n = this->n * rhs.n;
    this->d = this->d * rhs.d;
};
void Fraccion::operator/(const Fraccion &rhs){
    this->n = this->n * rhs.d;
    this->d = this->d * rhs.n;
};
string Fraccion::operator~() {
    stringstream s("", ios_base::app | ios_base::out);
    s<<"La fraccion es: " << n << "/" << d << endl;
    return s.str();
};
```

4.4. Código Main

`main.cpp`:

```
#include <iostream>

#include "polinomio_divi_multi.h"
#include "calculadora.h"
#include "fraccion.h"
#include "polinomio.h"


using namespace std;


int main (){


    cout <<"Probando calculdadora con doubles: " <<endl;

    Calculadora<double> cal1;
    double a=2.455;
    double b=6.1;

    double f = cal1.add(a,b);
    double &x = f;
    cout  << "a: "<< a  << " " << "b: "<< b << endl;
    cout<<"a+b" <<endl;
    cal1.print(x);

    double f2 = cal1.sub(a,b);
    double &x2 = f2;
    cout  << "a: "<< a  << " " << "b: "<< b << endl;
    cout<<"a-b" <<endl;
```

```
31      cal1.print(x2);
32
33      double f3 = cal1.mul(a,b);
34      double &x3 = f3;
35      cout  << "a: "<< a  << " " << "b: "<< b << endl;
36      cout<<"a*b" <<endl;
37      cal1.print(x3);
38
39      double f4 = cal1.div(a,b);
40      double &x4 = f4;
41      cout  << "a: "<< a  << " " << "b: "<< b << endl;
42      cout<<"a/b" <<endl;
43      cal1.print(x4);
44
45      cout <<"Haciendo operaciones con fracciones: " <<endl;
46      Fraccion fracc1(1,2);
47      Fraccion fracc2(2,3);
48      cout << "Fraccion A " <<~fracc1 <<endl;
49      cout << "Fraccion B " <<~fracc2 <<endl;
50      fracc1+fracc2;
51      cout << "C = A + B --" << ~fracc1<< endl;
52      fracc1-fracc2;
53      cout << "D = C - B --" << ~fracc1<< endl;
54      fracc1*fracc2;
55      cout << "E = D * B --" << ~fracc1<< endl;
56      fracc1/fracc2;
57      cout << "F = E / B --" << ~fracc1<< endl;
58
59
60
61      cout<<"Suma y resta de polinomios: " <<endl;
62
63      int* arr1 = new int[5];
64      for( int i =0; i<5 ; i++){
65          arr1[i]=i;
66      }
67
68      int* arr2 = new int[5];
69      for( int i =0; i<5 ; i++){
70          arr2[i]=i+2;
71      }
72
73      Polinomio pol1(5, arr1);
74      Polinomio pol2(5,arr2);
75      cout << "Polinomio 1: "<<~pol1;
76      cout << "Polinomio 2: " <<~pol2;
77      pol1+pol2;
78      cout << "Polinomio 1 + Polinomio 2: " <<~pol1<< endl;
79      cout << "Polinomio 1: "<<~pol1;
80      cout << "Polinomio 2: " <<~pol2;
81
82      pol1-pol2;
83      cout << "Polinomio 1 - Polinomio 2: " <<~pol1<< endl;
84
85
86
87
88
89      polinomio_divi_multi A(5, 7), B(3, 4), C(2), D(A), X, Y;
```

```
    polinomio_divi_multi A2, B2, Z1, Z2;
    cout << "      " << endl;
    cout << "Se define el polinomio A = 6x^4 -8x^3 +17x -9 " << endl;
    cout << "      " << endl;
    A.setCoeff(6, 4);
    A.setCoeff(17, 1);
    A.setCoeff(-8, 3);
    A.setCoeff(-9, 0);

    cout << "Se define B = 3x^6 +10x^3" << endl;
    cout << "      " << endl;
    B.setCoeff(3, 6);
    B.setCoeff(10, 3);


    cout << "Calculando  D = A * B " << endl;
    D = A * B ;
    cout << "      " << endl;
    cout << "Resultado D =" << D << endl;
    cout << "      " << endl;



    cout << "Se define B2 = -x^4 -3x^3" << endl;
    cout << "      " << endl;
    B2.setCoeff(-1, 4);
    B2.setCoeff(-3, 3);


    cout << "Se define el polinomio A2 = x^3 +3x^2" << endl;
    cout << "      " << endl;
    A2.setCoeff(1, 3);
    A2.setCoeff(3, 2);
    A2.setCoeff(0, 3);
    A2.setCoeff(0, 0);

    cout << "Calculando  X = B2 / A2 " << endl;
    X = A / B;

    cout << "      " << endl;
    cout << "Resultado  verificado en una calculadora online da: X = -1x y nuestro
    resultado calculado es: " << X << endl;
    cout << "      " << endl;




    return 0;
}
```

5. Consideraciones

1. Equipos de 2 o 3 personas.
2. Genere un reporte en LaTeXque incluya el enunciado, el diagrama de

clases, sus conclusiones y, como anexo incluya su código fuente

3. Suba su código y documentación (doxygen, README, INSTALL) al GitLab respectivo de su grupo y el directorio del laboratorio.

4. Cada estudiante debe subir el reporte a Schoology.

5. Recuerde que por cada día tardío de entrega se le rebajaran puntos de acuerdo con la formula: 4^d donde $d > 1$ es la cantidad de días tardíos