# Reporte de Laboratorio 3

Isacc Gómez Sánchez- B32919
José Fernando González Salas - B43023

29 de septiembre de 2016

---

## Índice



---

## 1. Introducción

En el siguiente reporte, se presenta un programa en el cual se utiliza parte de la programación genérica incluída en las funciones de C++, en donde se puede construir una plantilla para definir clases y utilizarla con cualquier tipo de dato. Además de la utilización de clases derivadas y sobrecarga de operadores para diferentes objetos de distinto tipo.

### 1.1. Objetivos

- Escribir un programa utilice clases bases y clases derivadas en C++
- Sobrecargar el operador «˜» para realizar funciones específicas de impresión y los operadores aritméticos para suma de fracciones, matrices y polinomios.
- Familiarizarse con las plantillas de C++.

# 2. Código

## 2.1. Plantilla de Calculadora

En este archivo encontramos la definición de la clase Calculadora por medio de una plantilla (template).

```
// Calculadora.h
#ifndef CALCULADORA_H
#define CALCULADORA_H


template <typename data>

class Calculadora {

public:

    Calculadora(){};
    Calculadora(const Calculadora& orig){};
    virtual ~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 &d){
        ~d;
    }
};
```

## 2.2. Archivo Fraccion.h

Archivo de cabecera de la clase Fracción, donde se definen sus métodos y atributos propios.

```
// Fraccion.h

#ifndef FRACCIONES_H
#define FRACCIONES_H

#include <iostream>
using namespace std;

class Fracciones {
public:
    Fracciones();
    Fracciones(int numerador, int denominador);
    Fracciones(const Fracciones& orig);
    virtual ~Fracciones();

    int numerador;
    int denominador;

    Fracciones operator+(const Fracciones &ELOTRO);
    Fracciones operator-(const Fracciones &ELOTRO);
    Fracciones operator*(const Fracciones &ELOTRO);
    Fracciones operator/(const Fracciones &ELOTRO);
    virtual void operator~();

};



#endif /* FRACCIONES_H */
```

## 2.3. Archivo Fraccion.cpp

Archivo donde se le da cuerpo a las funciones de la clase Fracción y se sobrecargan sus operadores.

```
// Fraccion.cpp
#include "Fracciones.h"

Fracciones::Fracciones() {
};

Fracciones::Fracciones(int numerador, int denominador) {

    this->numerador = numerador;
    this->denominador = denominador;

};

Fracciones::Fracciones(const Fracciones& orig) {
```

```
};

Fracciones::~Fracciones() {
};

Fracciones Fracciones::operator+(const Fracciones &ELOTRO) {
   Fracciones c;
    if (this->denominador == ELOTRO.denominador) {
        c.numerador = this->numerador + ELOTRO.numerador;
        c.denominador = this->denominador;
    } else {
        c.numerador = ELOTRO.denominador * this->numerador + this->denominador *
            ELOTRO.numerador;
        c.denominador = this->denominador * ELOTRO.denominador;
    }
    return c;

}

Fracciones Fracciones::operator-(const Fracciones &ELOTRO) {
    Fracciones c;
    if (this->denominador == ELOTRO.denominador) {
        c.numerador = this->numerador - ELOTRO.numerador;
        c.denominador = this->denominador;
    } else {
        c.numerador = ELOTRO.denominador * this->numerador - this->denominador *
            ELOTRO.numerador;
        c.denominador = this->denominador * ELOTRO.denominador;
    }
    return c;
}

Fracciones Fracciones::operator*(const Fracciones &ELOTRO) {
    Fracciones c;
    c.numerador = this->numerador * ELOTRO.numerador;
    c.denominador = this->denominador * ELOTRO.denominador;
    return c;
}

Fracciones Fracciones::operator/(const Fracciones &ELOTRO) {
    Fracciones c;
    c.numerador = this->numerador * ELOTRO.denominador;
    c.denominador = this->denominador * ELOTRO.numerador;
    return c;
}

void Fracciones::operator~() {
    cout << this->numerador << "/" << this->denominador << endl;
}
```

## 2.4. Archivo Matriz.h

Archivo de cabecera de la clase Matriz, donde se definen sus métodos y atributos propios.

```
// Matriz.h
#ifndef MATRIZ_H
#define MATRIZ_H

#include <iostream>
using namespace std;

class Matriz{
public:
    Matriz();
    Matriz(int columnas, int filas, double** arreglo);
    Matriz(const Matriz& orig);
    virtual ~Matriz();

    int columnas;
    int filas;
    double ** arreglo;


    Matriz operator+(Matriz b);
    Matriz operator-(Matriz b);
    Matriz operator*(Matriz b);
    Matriz operator/(Matriz b);
    virtual void operator~();
};
#endif /* MATRIZ_H */
```

## 2.5. Archivo Matriz.cpp

Archivo donde se le da cuerpo a las funciones de la clase Matriz y se sobrecargan sus operadores.

```
// Matriz.cpp
#include "Matriz.h"

Matriz::Matriz() {
    arreglo = new double*[filas];
    for (int i = 0; i < filas; ++i) {
        arreglo[i] = new int[columnas];
    };
}

Matriz::Matriz(int columnas, int filas, double** arreglo) {
    this->columnas = columnas;
    this->filas = filas;
    this->arreglo = arreglo;
}

Matriz::Matriz(const Matriz& orig) {
}

Matriz::~Matriz() {
    for (int i = 0; i < this->filas; ++i) {
```

```
        delete [] arreglo[i];
    }
    delete [] arreglo;
}

Matriz Matriz::operator + (Matriz b) {
    Matriz c;
    c.columnas = this->columnas;
    c.filas = this->filas;
    for (int i; i <= 7; i++) {
        for (int j; j <= 7; j++) {
            c.arreglo[i][j] = this->arreglo[i][j] + b.arreglo[i][j];
        }
    }
    return (c);
}

Matriz Matriz::operator - (Matriz b) {
    Matriz c;
    c.columnas = this->columnas;
    c.filas = this->filas;
    for (int i; i <= 7; i++) {
        for (int j; j <= 7; j++) {
            c.arreglo[i][j] = this->arreglo[i][j] - b.arreglo[i][j];
        }
    }
    return (c);
}

Matriz Matriz::operator*(Matriz b) {
    Matriz c;
    c.columnas = this->columnas;
    c.filas = this->filas;
    for (int i; i <= 7; i++) {
        for (int j; j <= 7; j++) {
            for (int k; k <= 7; k++)
                c.arreglo[i][j] = this->arreglo[i][k] += b.arreglo[k][j];
        }
    }
    return (c);
}

Matriz Matriz::operator / (Matriz b) {
    while (this->filas != b.filas || this->columnas != b.columnas) {
        cout << "Introduzca una matriz cuadrada";
    }
    Matriz c;
    c.columnas = this->columnas;
    c.filas = this->filas;
    for (int i; i <= 7; i++) {
        for (int j; j <= 7; j++) {
            c.arreglo[i][j] = this->arreglo[i][j] / b.arreglo[i][j];
        }
    }
```

```
    return (c);

}

void Matriz::operator~() {
    for (int i; i <= 7; i++) {
        cout << endl;
        for (int j; j <= 7; j++) {
            cout << this->arreglo[i][j];
        }
    }
}
```

## 2.6. Archivo Polinomios.h

Archivo de cabecera de la clase Polinomios, donde se definen sus métodos y atributos propios.

```
// Polinomios.h
#ifndef POLINOMIOS_H
#define POLINOMIOS_H

#include <string>

using namespace std;

class Polinomios {
public:
    Polinomios();
    Polinomios(int base, double* coeficientes);
    Polinomios(const Polinomios& orig);
    virtual ~Polinomios();

    int base;
    double* coeficientes;

    Polinomios operator+(const Polinomios &ELOTRO);
    Polinomios operator-(const Polinomios &ELOTRO);
    Polinomios operator*(const Polinomios &ELOTRO);
    Polinomios operator /(const Polinomios &ELOTRO);
    void operator~();

private:
    double resultado[];

};
```

## 2.7. Archivo Polinomios.cpp

Archivo donde se le da cuerpo a las funciones de la clase Polinomios y se sobrecargan sus operadores.

```
// Polinomios.cpp
```

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

Polinomios::Polinomios() {

    this -> base = 1;
    this -> coeficientes = 0;

};

Polinomios::Polinomios(int base, double* coeficientes) {

    this -> base = base;
    this -> coeficientes = coeficientes;

};

Polinomios::Polinomios(const Polinomios& orig) {
}

Polinomios::~Polinomios(){};

Polinomios Polinomios::operator + (const Polinomios &ELOTRO){

    int a;

    if (this->base < ELOTRO.base) {

        a = ELOTRO.base;

    }

    else {

        a = this -> base;

    }

    double *res = new double [a];

    for (int i = 0; i < a; i++) {

    res[i] = ELOTRO.coeficientes[i] + coeficientes[i];

    }

    Polinomios resultado(a, res);

    delete [] res;

    return resultado;

}
```

```
Polinomios Polinomios::operator - (const Polinomios &ELOTRO){
    int a;
    if (this->base < ELOTRO.base) {
        a = ELOTRO.base;
    }
    else {
        a = this -> base;
    }
    double *res = new double [a];
    for (int i = 0; i < a; i++) {
    cout << coeficientes[i] << " " << ELOTRO.coeficientes[i] << " " << res[i] << endl;
    res[i] = coeficientes[i] - ELOTRO.coeficientes[i];
    }
    Polinomios resultado(a, res);
    delete [] res;
    return resultado;
}
Polinomios Polinomios::operator*(const Polinomios &ELOTRO){
    int a = this -> base;
    int b = ELOTRO.base;
    double *res = new double [a + b];
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
            res[i + j] = ELOTRO.coeficientes[i] * coeficientes[j];
        }
    }
    Polinomios resultado(a + b, res);
```

```
    delete [] res;

    return resultado;

}

Polinomios Polinomios::operator /(const Polinomios &ELOTRO)
{
   int i, dd, dq, dN, base2;

        dN = this -> base;
        base2 = ELOTRO.base;

   dq = dN - base2;

   double *N = new double [dN];

        for ( i = 0; i < dN; i++ ) {

         N[i] = this -> coeficientes[i];

        }

        double *D = new double [base2];

         for ( i = 1; i < dN; i++ ) {

         D[i] = ELOTRO.coeficientes[i];


        }

   double *d = new double [dN];

   double *q = new double [dq];

        for( i = 0 ; i < dq; i++ ) {
      q[i] = 0;

   }

   if( base2 < 0) {
      cout << "El grado debe ser mayor a 0.";
   }


   if( dN >= base2 ) {
      while(dN >= base2) {

         for( i = 0 ; i < dN; i++ ) {
            d[i] = 0;

         }
         for( i = 0 ; i < base2; i++ ) {
```

```
            d[i + dN - base2] = D[i];


         }
         dd = dN - 1;
                 q[dN - base2] = N[dN - 1]/d[dd];


         for( i = 0 ; i < dq; i++ ) {
            d[i] = d[i] * q[dN - base2 - 1];

         }



         for( i = 0 ; i < dN; i++ ) {
            N[i] = N[i] - d[i];

         }
         dN --;

      }

   }

   Polinomios resultado(dq, q);
        cout << dq << "        " << q[dq] << endl;

        delete [] N;
        delete [] D;
        delete [] d;
        delete [] q;

        return resultado;

}

void Polinomios::operator~(){

   for(int i = 0; i < base; i++) {

        cout << coeficientes[i] << "x" << i << " ";

    };

    cout << endl;

}
```

## 2.8. Archivo main.cpp

Archivo principal del programa en ejecución.

```
// main.cpp

#include <cstdlib>
#include <string>
#include <ctime>

#include "Calculadora.h"
#include "Fracciones.h"
#include "Matriz.h"
#include "Polinomios.h"

using namespace std;

int main(int argc, char** argv) {

    Calculadora<Fracciones> fr;

    Fracciones fa;
    fa.numerador=3;
    fa.denominador=4;

    Fracciones fb;
    fb.numerador=2;
    fb.denominador=4;

    Fracciones f;

    f=fr.add(fa, fb);
    fr.print(f);


    return 0;

}
```

## 2.9. Archivo Makefile

Este archivo se genera para compilar, ejecutar y eliminar los archivos de compilaciones anteriores.

```
// Makefile
compilar:
   g++ -o main.c++ main.cpp Fraccion.cpp Matriz.cpp Polinomios.cpp

borrar:
   rm main.c++

ejecutar: compilar
   ./main.c++
```

# 3. Diagrama UML

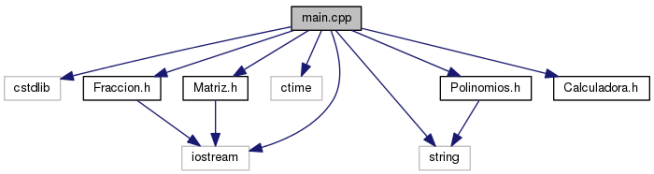


Figura 1: Se muestra el diagrama de jerarquía UML del programa.

# 4. Conclusiones

- Se escribe un programa con clases vases y derivadas en C++ con éxito.
- Se logra sobrecargar los operadores indicados para realizar las distintas operaciones especificadas, tanto de impresión como de operaciones aritméticas de distintos tipos de datos.
- Se familiriariza con las plantillas o templates facilitados por C++.