Reporte de la Tarea 2

Luis Diego Fernández Coto - B22492

24 de enero de 2017

Índice

---

1. Enunciado

EIE
Escuela de Ingeniería Eléctrica

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

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

Sobrecarga de operadores: polinomios

M. Sc. Ricardo Román Brenes - `ricardo.roman@ucr.ac.cr`

III-2016

Tabla de contenidos

---

1. Enunciado

Implemente una clase en C++ que modele polinomios, utilizando sobrecarga de operadores que efectúe las cuatro operaciones básicas algebraicas:

- `P& operator+(const P& rhs); //suma`
- `P& operator-(const P& rhs); //resta`
- `P& operator*(const P& rhs); //multiplicación`
- `P& operator/(const P& rhs); //división`

Para la división, utilice el método de la división sintética con sus restricciones.

Además programe los métodos necesarios para que estos operaciones funcionen: impresión, construcción de copia, asignaciones, etc.

Haga un programa de prueba en donde se creen objetos e interactúen entre ellos.

Escriba también un Makefile con al menos tres reglas:

1. build: para compilar su programa.
2. run: para ejecutar el programa de prueba.
3. clean: para eliminar ejecutables y archivos intermedios.

2. Consideraciones

- Trabajo individual.
- Genere un reporte en LaTeX que incluya al menos el enunciado, la solución propuesta, su código y sus conclusiones.
- Suba su código con documentación interna al GitHub respectivo de su grupo y el directorio del laboratorio.
- Cada estudiante debe subir el reporte a Schoology. (`https://app.schoology.com/assignment/947398948/`).
- Recuerde que por cada día tardío de entrega se le rebajaran puntos de acuerdo con la formula: 3^d, donde $d > 1$ es la cantidad de días naturales tardíos.

2. Solución propuesta

En esta tarea se nos propone crear una clase que logre modelar un polinomio de orden n y hacer con él las 4 operaciones básicas (suma, resta, multiplicación y división) con otros polinomios. Para lograr esto primero es necesario crear una clase Polinomio. Esta clase va a tener 2 atributos fundamentales:

- double* data: este nos va a servir luego para almacenar los valores de cada coeficiente del polinomio. Es un puntero a un double, que en el costructor lo hacemos "*apuntar*" a un vector de datos tipo double.
- int grado: este va a ser un valor entero que nos va a indicar que grado tiene el polinomio.

Luego de esto se procede a implementar los constructores y el destructor de esta clase, en este caso vamos a tener 4, que son los siguientes:

- Constructor por defecto: en este caso el constructor por defecto tiene un cuerpo vacío y no hace "*nada*".
- Constructor por conversión: este recibe como argumentos el grado del polinomio que se quiere crear y recibe un vector de doubles que va a corresponder a los valores de los coeficientes del polinomio. Con este vector de doubles se crea en memoria dinámica un vector (este va a ser apuntado por el atributo data) donde se van a meter los valores de este vector recibido y esots van a ser los valores de los coeficientes del polinomio. Es importante mencionar que el índice de este vector corresponde al valor de dicho coeficiente del polinomio, por ejemplo si la posición 2 del vector tiene un 5, eso equivale al término $5x^2$ en el polinomio. Esta convención se sigue en el diseño de todo el código.
- Constructor por copia: este lo que hace es que recibe una referencia de un polinomio y crea otro polinomio que es una copia, valga la redundancia, del recibido.
- Destructor: básicamente lo único que hace es liberar la memoria dinámica utilizada al construir cada polinomio.

Con esto ya podemos aunque sea en un main crear objetos de tipo polinomio, sin embargo, no podemos observar que hacen, ni siquiera si funcionan. Es por esto que se implementa un método que imprime el polinomio, este método simplemente imprime los valores del vector data y dependiendo de la posción les pone al lado un x^n donde n corresponde al respectivo índice del vector data. Esto se puede observar en la figura 1.

```
P1:
7x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x^1 + 20
```

Figura 1: Impresión del polinomio P1.

Ahora nos corresponde implementar las operaciones básicas. Para esto se hace una sobrecarga de los operadores de estas operaciones. Se pude observar la definición de estas en la figura 2.

```
//sobrecarga de diferentes operadores
Polinomio operator=(const Polinomio& other);
Polinomio operator+(const Polinomio& other);
Polinomio operator-(const Polinomio& other);
Polinomio operator*(const Polinomio& other);
Polinomio operator/(const Polinomio& other);
```

Figura 2: Definición de la sobrecarga de los operadores.

A continuación explicamos que se hace en cada una de ellas:

- Sobrecarga del operador de asignación: este operador es binario por lo que recibe la referencia de un polinomio y se la tiene que asignar a otro polinomio que recibe de manera implícita como un puntero a this.
- Sobrecarga del operador de suma: en este caso lo que se hace es que reciben 2 polinomios, uno como argumento y otro implícito (como un puntero a this) y se encarga de sumar ambos polinomios. Básicamente lo que hace la suma es que toma el vector de datos de cada polinomio y suma los datos que están en la misma posición en cada vector. Estos resultados de las sumas los almacena en un vector especial creado para dicho fin y retorna un polinomio creado con este vector.
- Sobrecarga del operador de resta: la sobrecarga de este operador realiza exacatamente lo mismo que la sobrecarga del operador suma, solo que en lugar de sumar los valores de los vectores los resta, evidentemente.
- Sobrecarga del operador de multiplicación: en este caso lo que se hace es implementar un doble ciclo for que multiplique cada posicion de uno de los vectores de datos por todas las posiciones del otro vector de datos, almacenando los resultados en un nuevo vector. Este nuevo vector va a tener un tamaño igual a la suma de los grados de los polinomios que se están multiplicando y cada multiplicacion se guarda en la posicion equivalente a la suma de los índices que se están multiplicando.
- Sobrecarga del operador de división: la implementación de este método si fue un poco más complicada. Lo que se realizó fue que se siguió el algoritmo de la división larga de polinomios. Esta lo que nos dice es que se siguen 3 pasos fundamentales y se respiten hasta que no se pueda continuar. estos pasos son los siguietnes:

  1. Se divide el primer término del numerador por el primer término del denominador y este va a formar parte del polinomio cociente.
  2. Se multiplica el denominador por el resultado del paso anterior.
  3. Se resta el resultado del paso anterior al numerador y se crea un "*nuevo*" polinomio.

  Para realizar esto se ocupa entonces el polinomio (más que todo el vector de datos de cada uno) numerador (this), el polinomio denominador (argumento) y la creación de 2 vectores auxiliares para almacenar los resultados de los puntos 2 y 3. Este procedimiento se itera una cantidad de veces igual al grado del denominador o divisor. Al finalizar, la solución va a ser igual a la suma del cociente resultante más el residuo. Importante destacar que en este caso y para esta implementación, el grado del polinomio dividendo no puede ser menor al grado del divisor, por lo tanto para dicho caso se mostrará un mensaje de error al usuario.

3. Código

3.1. main.cpp

```
#include <iostream>

#include "Polinomio.h"

using namespace std;

int main() {
    //se crean los vectores que van a tener los valores de los polinomios
    //el indice dentro el vector corresponde al coeficiente dentro del polinomio
    double vector0[] = {10, 1, 2, 3, 4, 5};
    double vector1[] = {20, 1, 2, 3, 4, 5, 6, 7};
    double vector2[] = {13, -8, -11, 15, 3, 9, -9, 8, 10, 1};
    double vector3[] = {-10, -3, 1};
    double vector4[] = {2, 1};
    double vector5[] = {-9, 6, 0, 0, 2, 0, 1};
    double vector6[] = {3, 0, 0, 1};

    //se crean 7 polinomio
    Polinomio* p0 = new Polinomio(5, vector0);
    Polinomio* p1 = new Polinomio(7, vector1);
    Polinomio* p2 = new Polinomio(9, vector2);
    Polinomio* p3 = new Polinomio(2, vector3);
    Polinomio* p4 = new Polinomio(1, vector4);
    Polinomio* p5 = new Polinomio(6, vector5);
    Polinomio* p6 = new Polinomio(3, vector6);

    //se crea un polinomio utilizando el constructor por copia
    Polinomio copia = *p0;

    //se procede a crear ciertos polinomios con el constructor por defecto
    //se les pone nombres significativos que indican el resultado que va a ser asig
    Polinomio* sumaP0P1 = new Polinomio();
    Polinomio* sumaP0P2 = new Polinomio();
    Polinomio* restaP0P1 = new Polinomio();
    Polinomio* restaP1P0 = new Polinomio();
    Polinomio* restaP1P2 = new Polinomio();
    Polinomio* multP0P1 = new Polinomio();
    Polinomio* multP0P2 = new Polinomio();
    Polinomio* multP1P2 = new Polinomio();
    Polinomio* divP3P4 = new Polinomio();
    Polinomio* divP5P6 = new Polinomio();

    //se imprimen los 7 polinomios creados anteriormente
    cout << "P0: " << endl;
```

```
p0->print();

cout << "\nP1: " << endl;
p1->print();

cout << "\nP2: " << endl;
p2->print();

cout << "\nP3: " << endl;
p3->print();

cout << "\nP4: " << endl;
p4->print();

cout << "\nP5: " << endl;
p5->print();

cout << "\nP6: " << endl;
p6->print();

//se imprime el polinomio que resulto del constructor por copia
cout << "\nCopia de P0: " << endl;
copia.print();

//se hacen las distintas operaciones, se asigan al polinomio correspondiente y
*sumaP0P1 = *p0 + *p1;
cout << "\nP0 + P1: " << endl;
sumaP0P1->print();

*sumaP0P2= *p0 + *p2;
cout << "\nP0 + P2: " << endl;
sumaP0P2->print();

*restaP0P1 = *p0 - *p1;
cout << "\nP0 - P1: " << endl;
restaP0P1->print();

*restaP1P0 = *p1 - *p0;
cout << "\nP1 - P0: " << endl;
restaP1P0->print();

*restaP1P2 = *p1 - *p2;
cout << "\nP1 - P2: " << endl;
restaP1P2->print();

*multP0P1 = *p0 * *p1;
cout << "\nP0 * P1: " << endl;
multP0P1->print();
```

```
    *multP0P2 = *p0 * *p2;
    cout << "\nP0 * P2: " << endl;
    multP0P2->print();

    *multP1P2 = *p1 * *p2;
    cout << "\nP1 * P2: " << endl;
    multP1P2->print();

    *divP3P4 = *p3 / *p4;
    cout << "\nP3 / P4: " << endl;
    divP3P4->print();

    *divP5P6 = *p5 / *p6;
    cout << "\nP5 / P6: " << endl;
    divP5P6->print();

    //se finaliza el programa
    return 0;
}
```

3.2. Polinomio.h

```
#ifndef POLINOMIO_H
#define POLINOMIO_H

#include <iostream>

using namespace std;

class Polinomio {
private:
    double* data;
    int grado;

public:
    ///constructor por defecto
    Polinomio();

    ///constructor por conversion
    Polinomio(int grado, double* data);

    ///constructor por copia
    Polinomio(const Polinomio& other);

    ///destructor
    ~Polinomio();
```

```
    //sobrecarga de diferentes operadores
    Polinomio operator=(const Polinomio& other);
    Polinomio operator+(const Polinomio& other);
    Polinomio operator-(const Polinomio& other);
    Polinomio operator*(const Polinomio& other);
    Polinomio operator/(const Polinomio& other);

    void print ();
};

#endif // POLINOMIO_H
```

3.3. Polinomio.cpp

```
#include "Polinomio.h"

Polinomio::Polinomio() {

}

Polinomio::Polinomio(int grado, double* data) {
    this->grado = grado;
    this->data = new double [this->grado + 1];
    for (int index = 0; index <= this->grado; ++index)
        this->data[index] = data[index];
}

Polinomio::Polinomio(const Polinomio& other) {
    this->grado = other.grado;
    this->data = new double [this->grado + 1];
    for (int index = 0; index <= this->grado; ++index)
        this->data[index] = other.data[index];
}

Polinomio::~Polinomio() {
    delete [] this->data;
}

Polinomio Polinomio::operator=(const Polinomio& other) {
    if (this == &other)
        return *this;

    this->grado = other.grado;
    this->data = new double [this->grado + 1];
    for (int index = 0; index <= this->grado; ++index)
        this->data[index] = other.data[index];

    return *this;
```

```
}

Polinomio Polinomio::operator+(const Polinomio& other) {
    int nuevoGrado = 0;
    if (this->grado > other.grado)
        nuevoGrado = this->grado;
    else
        nuevoGrado = other.grado;

    double resultVector[nuevoGrado + 1];

    for (int index = 0; index < nuevoGrado + 1; ++index) {
        if (this->grado >= index && other.grado >= index)
            resultVector[index] = this->data[index] + other.data[index];
        else {
            if (this->grado < index)
                resultVector[index] = other.data[index];
            else
                resultVector[index] = this->data[index];
        }
    }

    Polinomio* result = new Polinomio(nuevoGrado, resultVector);
    return *result;
}

Polinomio Polinomio::operator-(const Polinomio& other) {
    int nuevoGrado = 0;
    if (this->grado > other.grado)
        nuevoGrado = this->grado;
    else
        nuevoGrado = other.grado;

    double resultVector[nuevoGrado + 1];

    for (int index = 0; index < nuevoGrado + 1; ++index) {
        if (this->grado >= index && other.grado >= index)
            resultVector[index] = this->data[index] - other.data[index];
        else {
            if (this->grado < index)
                resultVector[index] = other.data[index];
            else
                resultVector[index] = this->data[index];
        }
    }

    Polinomio* result = new Polinomio(nuevoGrado, resultVector);
    return *result;
```

```
}

Polinomio Polinomio::operator*(const Polinomio& other) {
    int nuevoGrado = this->grado + other.grado;
    double resultData [nuevoGrado + 1];

    for (int index = 0; index <= nuevoGrado; ++index)
        resultData[index] = 0.0;

    for (int outIndex = 0; outIndex <= this->grado; ++outIndex) {
        for (int inIndex = 0; inIndex <= other.grado; ++inIndex) {
            resultData[outIndex + inIndex] = (this->data[outIndex] * other.data[inI
            + resultData[outIndex + inIndex];
        }
    }

    Polinomio* result = new Polinomio(nuevoGrado, resultData);
    return *result;
}

Polinomio Polinomio::operator/(const Polinomio& other) {
    int grado = other.grado;
    double quotient [grado + 1];
    double remainder [grado + 1];

    double temp [this->grado + 1];
    for (int index = 0; index <= this->grado; ++index)
        temp[index] = this->data[index];

    for (int dividendIndex = 0; dividendIndex <= other.grado; ++dividendIndex) {
        quotient[grado - dividendIndex] = temp[this->grado - dividendIndex] / other

        for (int remainderIndex = 0; remainderIndex <= other.grado; ++remainderInde
            remainder[grado - remainderIndex] = quotient[grado - dividendIndex] * o
        }

        for (int restIndex = 0; restIndex <= other.grado; ++restIndex) {
            temp[this->grado - restIndex - dividendIndex] = temp[this->grado - rest
        }
    }

    Polinomio* semiResult = new Polinomio(grado, quotient);
    Polinomio* tempo = new Polinomio(this->grado, temp);

    return *semiResult + *tempo;
}

void Polinomio::print() {
```

```
    //cout << "El polinomio esta en: " << this << endl;
    for (int index = 0; index <= this->grado; ++index) {
        if (index != grado) {
            if (this->data[grado - index] == 1)
                cout << "x^" << this->grado - index << " + ";
            else
            cout << this->data[grado - index] << "x^" << this->grado - index << " +
        }
        else
            cout << this->data[grado - index] << endl;
    }
}
```

4. Resultados

Para probar el correcto funcionamiento del programa se creó un main donde se crearon 7 polinomios con el constructor por conversión como se observa en la figura 3 y se imprimieron como se observa en la figura 4.

```
//se crean los vectores que van a tener los valores de los polinomios
//el indice dentro el vector corresponde al coeficiente dentro del polinomio
double vector0[] = {10, 1, 2, 3, 4, 5};
double vector1[] = {20, 1, 2, 3, 4, 5, 6, 7};
double vector2[] = {13, -8, -11, 15, 3, 9, -9, 8, 10, 1};
double vector3[] = {-10, -3, 1};
double vector4[] = {2, 1};
double vector5[] = {-9, 6, 0, 0, 2, 0, 1};
double vector6[] = {3, 0, 0, 1};

//se crean 7 polinomio
Polinomio* p0 = new Polinomio(5, vector0);
Polinomio* p1 = new Polinomio(7, vector1);
Polinomio* p2 = new Polinomio(9, vector2);
Polinomio* p3 = new Polinomio(2, vector3);
Polinomio* p4 = new Polinomio(1, vector4);
Polinomio* p5 = new Polinomio(6, vector5);
Polinomio* p6 = new Polinomio(3, vector6);
```

Figura 3: Creación de 7 polinomios distintos.

```
P0:
5x^5 + 4x^4 + 3x^3 + 2x^2 + x^1 + 10

P1:
7x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x^1 + 20

P2:
x^9 + 10x^8 + 8x^7 + -9x^6 + 9x^5 + 3x^4 + 15x^3 + -11x^2 + -8x^1 + 13

P3:
x^2 + -3x^1 + -10

P4:
x^1 + 2

P5:
x^6 + 0x^5 + 2x^4 + 0x^3 + 0x^2 + 6x^1 + -9

P6:
x^3 + 0x^2 + 0x^1 + 3
```

Figura 4: Creación e impresión de 7 polinomios distintos.

Luego para probar la sobrecarga del operador de asignación se creó un polinomio por defecto y se igualó al polinomio P0 como se observa en la figura 5, luego se imprimió como se observa en la figura 6.

```
//se crea un polinomio utilizando el constructor por copia
Polinomio copia = *p0;
```

Figura 5: Copia del polinomio P0.

```
Copia de P0:
5x^5 + 4x^4 + 3x^3 + 2x^2 + x^1 + 10
```

Figura 6: Impresión de la copia del polinomio P0.

Por último se realizan ciertas operaciones y se imprime su resultado como se observa en la figura 7. De esta manera también se comprueba que la sobrecarga de los 4 operadores se realizó de manera correcta y producen el realizado esperado que fue confirmado contra los resultados de `http://www.wolframalpha.com/`

```
P0 + P1:
7x^7 + 6x^6 + 10x^5 + 8x^4 + 6x^3 + 4x^2 + 2x^1 + 30

P0 + P2:
x^9 + 10x^8 + 8x^7 + -9x^6 + 14x^5 + 7x^4 + 18x^3 + -9x^2 + -7x^1 + 23

P0 - P1:
7x^7 + 6x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 0x^1 + -10

P1 - P0:
7x^7 + 6x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 0x^1 + 10

P1 - P2:
x^9 + 10x^8 + -1x^7 + 15x^6 + -4x^5 + x^4 + -12x^3 + 13x^2 + 9x^1 + 7

P0 * P1:
35x^12 + 58x^11 + 70x^10 + 72x^9 + 65x^8 + 120x^7 + 95x^6 + 170x^5 + 130x^4 + 94
x^3 + 61x^2 + 30x^1 + 200

P0 * P2:
5x^14 + 54x^13 + 83x^12 + 19x^11 + 54x^10 + 60x^9 + 204x^8 + 103x^7 + -114x^6 +
123x^5 + 51x^4 + 162x^3 + -92x^2 + -67x^1 + 130

P1 * P2:
7x^16 + 76x^15 + 121x^14 + 39x^13 + 92x^12 + 94x^11 + 177x^10 + 83x^9 + 182x^8 +
 226x^7 + -126x^6 + 213x^5 + 81x^4 + 312x^3 + -202x^2 + -147x^1 + 260

P3 / P4:
0x^2 + x^1 + -5

P5 / P6:
0x^6 + 0x^5 + 0x^4 + x^3 + 0x^2 + 2x^1 + -3
```

Figura 7: Distintas operaciones de polinomios.

5. Conclusiones

- La implementación de la clase Polinomio y sus métodos para realizar operaciones entre ellos fue exitosa como se observa en la sección de resultados.
- Los 3 constructores (por defecto, de conversiön y de copia) así como el destructor funcionan apropiadamente.
- La sobrecarga de los operadores logra realizar lo que cada operador debería hacer sobre un Polinomio.