Reporte de Laboratorio 3
Programación genérica en C++

Dunia Barahona - B40806

28 de septiembre de 2016

Índice

---

1. Código

1.1. Clase emplantillada: Calculadora

Calculadora.h

```
#ifndef CALCULADORA_H
#define CALCULADORA_H
#include "Fraccion.h"
#include "Polinomio.h"
#include "Matriz.h"

template <typename data> //data puede ser Fraccion, polinomio o matriz
class Calculadora {
public:
        Calculadora() { //Constructor
        }
        ~Calculadora() { //Destructor
        }
        data add(data d1, const data d2) {
                data d= d1+d2;
                return d;
        }
```

```
        data sub(data d1, const data d2) {
                data d= d1-d2;
                return d;
        }
        data mul(data d1, const data d2) {
                data d= d1*d2;
                return d;
        }
        data div(data d1, const data d2) {
                data d= d1/d2;
                return d;
        }
        void print(data d) {
                ~d;
        }
};
#endif /* CALCULADORA_H */
```

1.2. Clase Fraccion

Fraccion.h

```
#ifndef FRACCION_H
#define FRACCION_H

#include <iostream>
#include "string"

using namespace std;

class Fraccion {
public:
        double num; //Numerador
        double den; //Denominador

        Fraccion();  //Constructor
        Fraccion(double n, double d);
        ~Fraccion(); //Destructor

        Fraccion operator+(const Fraccion f2);
        Fraccion operator-(const Fraccion f2);
        Fraccion operator*(const Fraccion f2);
        Fraccion operator/(const Fraccion f2);
        void operator~();
};
#endif /* FRACCION_H */
```

Fraccion.cpp

```
#include "Fraccion.h"

        Fraccion::Fraccion() {  //constructor
        }
        Fraccion::Fraccion(double num, double den) {
                this->num =num;
                this->den =den;
        }
        Fraccion::~Fraccion() { //Destructor
        }
        Fraccion Fraccion::operator+(const Fraccion f2) {
                Fraccion fadd;
                fadd.num= (this->num * f2.den)+(f2.num * this->den);
                fadd.den= (this->den * f2.den);
                return fadd;
        }
        Fraccion Fraccion::operator-(const Fraccion f2) {
                Fraccion fsub;
                fsub.num= (this->num * f2.den)-(f2.num * this->den);
                fsub.den= (this->den * f2.den);
                return fsub;
        }
        Fraccion Fraccion::operator*(const Fraccion f2) {
                Fraccion fmul;
                fmul.num= this->num * f2.num;
                fmul.den= this->den * f2.den;
                return fmul;
        }
        Fraccion Fraccion::operator/(const Fraccion f2) {
                Fraccion fdiv;
                fdiv.num= this->num * f2.den;
                fdiv.den= this->den * f2.num;
                return fdiv;
        }
        void Fraccion::operator~() {
                cout<< this->num<< "/"<< this->den<< endl;
        }
```

1.3. Clase Matriz

Matriz.h

```
#ifndef MATRIZ_H
#define MATRIZ_H

#include <cstdlib>
#include <iostream>
#include "string"

using namespace std;

class Matriz{
public:
        int m; //Filas de la matriz
        int n; //Columnas de la matriz
        double **matrix; //Coeficientes de la matriz

        Matriz();  //Constructor
        Matriz(int m, int n, double** matriz);
        ~Matriz(); //Destructor

        Matriz operator+(const Matriz f2);
        Matriz operator-(const Matriz f2);
        Matriz operator*(const Matriz f2);
        Matriz operator/(const Matriz f2);
        void operator~();
};
#endif /* MATRIZ_H */
```

Matriz.cpp

```
#include "Matriz.h"

Matriz::Matriz() { //Constructor simple
}
Matriz::Matriz(int m, int n, double** matrix) { //Constructor con atributos
        this->m =m;
        this->n =n;
        this->matrix =matrix;
}
Matriz::~Matriz() {     //Destructor
}
Matriz Matriz::operator+(const Matriz m2) {
        Matriz m1(this->m, this->n, this->matrix);
        Matriz madd(m,n,this->matrix);
        if(this->m != m2.m||this->n != m2.n){
```

```
                cout<<"Las dimensiones no coinciden"<<endl;
                return m1;
        }
        else{
        for(int i=0;i<m;i++){
        for(int j=0; j<n; j++){
                madd.matrix[i][j]=this->matrix[i][j]+m2.matrix[i][j];
        }
        }
        }
        return madd;
}
Matriz Matriz::operator-(const Matriz m2) {
        Matriz m1(this->m, this->n, this->matrix);
        Matriz madd(m,n,this->matrix);
        if(this->m != m2.m||this->n != m2.n){
                cout<<"Las dimensiones no coinciden"<<endl;
                return m1;
        }
        else{
                for(int i=0;i<m;i++){
                for(int j=0; j<n; j++){
                madd.matrix[i][j]=this->matrix[i][j]-m2.matrix[i][j];
                }
                }
        }
        return madd;
}
Matriz Matriz::operator*(const Matriz m2) {
        double **mat2 = (double **) malloc(sizeof(double *)*this->m);
        for(int i=0; i<m; i++){
                mat2[i] = (double *) malloc(sizeof(double)*m2.n);
        }
        for(int i=0;i<this->m;i++){
        for(int j=0; j<m2.n; j++){
                mat2[i][j]=0;
        }
        }
        Matriz mmul(this->m,m2.n,mat2);
        Matriz m1(this->m, this->n, this->matrix);
        if(this->n != m2.m){
                cout<< "Las matrices no son multiplicables" << endl;
                return m1;
        }
        else{
                for(int i=0;i<this->m;i++){
                for(int j=0; j<m2.n; j++){
                for(int k=0; k<this->n; k++){
```

```
                mmul.matrix[i][j]+=this->matrix[i][k]*m2.matrix[k][j];
                }
                }
                }
                return mmul;
        }
}
Matriz Matriz::operator/(const Matriz m2) {
        Matriz m1(this->m, this->n, this->matrix);
        Matriz madd(m,n,this->matrix);
        if(this->m != m2.m||this->n != m2.n){
                cout<<"Las dimensiones no coinciden"<<endl;
                return m1;
        }
        else{
        for(int i=0;i<m;i++){
        for(int j=0; j<n; j++){
        madd.matrix[i][j]=this->matrix[i][j]/m2.matrix[i][j];
        }
        }
        }
        return madd;
}
void Matriz::operator~() {
        for(int i=0;i<m;i++){
        for(int j=0; j<n; j++){
                cout<< this->matrix[i][j]<<"\t";
        }
        cout<<endl;
        }
}
```

1.4. Clase Polinomio

Polinmio.h

```
#ifndef POLINOMIO_H
#define POLINOMIO_H

#include <iostream>
#include "string"

using namespace std;

class Polinomio {
public:
        int tam;
```

```
        char var;       ///Variable del polinomio
        double* coef; //Coeficientes numÃ©ricos del polinomio

        Polinomio();
        Polinomio(int tam, char var, double* coef);
        ~Polinomio();

        Polinomio operator+(const Polinomio p2);
        Polinomio operator-(const Polinomio p2);
        Polinomio operator*(const Polinomio p2);
        Polinomio operator/(const Polinomio p2);
        void operator~();
};
#endif /* POLINOMIO_H */
```

Polinomio.cpp

```
#include "Polinomio.h"

Polinomio::Polinomio() { //Constructor simple
}
Polinomio::Polinomio(int tam, char var, double* coef) {
        this->tam= tam;
        this->var= var;
        double* temp= new double[tam];
        for (int i= 0; i< tam; i++)
        {
        temp[i]= coef[tam-1-i];
        }
        //les da vuelta, queda vector donde la posicion es el exponente
        this->coef= temp;
}
Polinomio::~Polinomio() { //Destructor
}
Polinomio Polinomio::operator+(Polinomio p2) {
        Polinomio pm, pM, padd;
        if (this->tam > p2.tam) {
                pm= p2;
                pM.tam= this->tam;
                pM.var= this->var;
                pM.coef= this->coef;
        }
        else {
                pM= p2;
                pm.tam= this->tam;
                pm.var= this->var;
                pm.coef= this->coef;
```

```
        }
        double* temp= new double[pM.tam];
        for (int i = 0; i < pm.tam; i++) {
                temp[i]= this->coef[i] + p2.coef[i];
        }
        for (int i = 0; i < (pM.tam-pm.tam); i++) {
                temp[pm.tam+i]= pM.coef[pm.tam+i];
        }
        padd.coef= temp;
        padd.tam= pM.tam;
        padd.var= this->var;
        return padd;
        delete[] temp;
}
Polinomio Polinomio::operator-(Polinomio p2) {
        Polinomio pm, pM, psub;
        if (this->tam > p2.tam) {
                pm= p2;
                pM.tam= this->tam;
                pM.var= this->var;
                pM.coef= this->coef;
        }
        else{
                pM= p2;
                pm.tam= this->tam;
                pm.var= this->var;
                pm.coef= this->coef;
        }
        double* temp= new double[pM.tam];
        for (int i = 0; i < pm.tam; i++) {
                temp[i]= this->coef[i] - p2.coef[i];
        }
        for (int i = 0; i < (pM.tam-pm.tam); i++) {
                if (this->tam > p2.tam) {
                        temp[pm.tam+i]= pM.coef[pm.tam+i];
                }
                else {
                temp[pm.tam+i]= (-1)*pM.coef[pm.tam+i];
                }
        }
        psub.coef= temp;
        psub.tam= pM.tam;
        psub.var= this->var;
        return psub;
        delete[] temp;
}
Polinomio Polinomio::operator*(Polinomio p2) {
        Polinomio pmul;
```

```
        pmul.tam= this->tam*p2.tam;
        pmul.var= this->var;
        int i=0, tp=0;
        double* temp= new double[pmul.tam];
        for (int a = 0; a < this->tam; a++) {
                for (int b = 0; b < p2.tam; b++) {
                        tp= this->coef[a] * p2.coef[b];
                        i= a+b;
                        temp[i]+= tp;
                }
        }
        pmul.coef= temp;
        return pmul;
        delete[] temp;
}
Polinomio Polinomio::operator/(Polinomio p2) {
        int tamdiv=(this->tam-p2.tam)+1, j=1;
        double* div= new double[tamdiv];
        double* cociente= new double[tamdiv];
        Polinomio pdiv, pb, pc, pd;
        pdiv.coef= div;
        pdiv.var= this->var;
        pdiv.tam= tamdiv;
        pb.coef= this->coef;
        pb.var= this->var;
        pb.tam= this->tam;
        for (int i =tamdiv-1; i >=0; i--) {
                for (int k = 0; k < tamdiv; k++) {
                        div[k]=0;
                }
        div[i]= pb.coef[pb.tam-j]/p2.coef[p2.tam-1];
        cociente[i]= div[i];
        pdiv.coef= div;
        pc= pdiv*p2;
        pd= pb-pc;
        pb.coef= pd.coef;
        j++;
        }
        pdiv.coef= cociente;
        return pdiv;
        delete[] div;
        delete[] cociente;
}
void Polinomio::operator~() {
        int cont=0;
        int a= this->tam-1;
        while (this->coef[a]==0) {
                cont++;
```

```
                a--;
        }
        for (int i= this->tam-1-cont; i>=0 ; i--) {
                if (i==0) {
                        if (this->coef[i]!=0) {
                                cout<<this->coef[i];
                        }
                }
                if (i>0) {
                if (this->coef[i]==-1) {
                        cout<<'-';
                }
                if (this->coef[i]!=0&&this->coef[i]!=1&&this->coef[i]!=-1) {
                        cout<<this->coef[i];
                }
                if (this->coef[i]!=0) {
                        cout<<this->var;
                }
                if ((i!=1)&&(this->coef[i]!=0)) {
                        cout<<'^'<<i;
                }
                if (this->coef[i-1]>0) {
                        cout<<'+';
                }
                }
                if (i==0) {
                        cout<<endl;
                }
        }
}
```

1.5. main

```
#include "Fraccion.h"
#include "Polinomio.h"
#include "Matriz.h"
#include "Calculadora.h"

int main(int argc, char** argv) {
        Calculadora<Fraccion>* c1= new Calculadora<Fraccion>();
        Calculadora<Matriz>* c2= new Calculadora<Matriz>();
        Calculadora<Polinomio>* c3= new Calculadora<Polinomio>();

        Fraccion f1= Fraccion (22.0, 7.0);
        Fraccion f2= Fraccion (40.0, 20.0);
        cout<<endl<<"Fraccion f1:\t";
        c1->print(f1);
```

```
cout<<"Fraccion f2:\t";
c1->print(f2);
cout<<"f1+f2=\t";
c1->print(c1->add(f1, f2));
cout<<"f1-f2=\t";
c1->print(c1->sub(f1, f2));
cout<<"f1*f2=\t";
c1->print(c1->mul(f1, f2));
cout<<"f1/f2=\t";
c1->print(c1->div(f1, f2));

double a[5] = {38, -65, 0, 0, 27};
double b[3] = {2, -5, 3};
Polinomio pa= Polinomio(5, 'x', a);
Polinomio pb= Polinomio(3, 'x', b);
cout<<endl<<"Polinomio a:\t";
c3->print(pa);
cout<<"Polinomio b:\t";
c3->print(pb);
cout<<"a+b=\t";
c3->print(c3->add(pa, pb));
cout<<"a-b=\t";
c3->print(c3->sub(pa, pb));
cout<<"a*b=\t";
c3->print(c3->mul(pa, pb));
cout<<"a/b=\t";
c3->print(c3->div(pa, pb));

double l=0;
int m=4;
int n=4;
double **mat1 = (double **) malloc(sizeof(double *)*m);
for(int i=0; i<m; i++){
        mat1[i] = (double *) malloc(sizeof(double)*n);
}
for(int i=0;i<m;i++){
        for(int j=0; j<n; j++){
                mat1[i][j]=2;
        }
}
double **mat2 = (double **) malloc(sizeof(double *)*m);
for(int i=0; i<m; i++){
        mat2[i] = (double *) malloc(sizeof(double)*n);
}
for(int i=0;i<m;i++){
        for(int j=0; j<n; j++){
                mat2[i][j]=3;
        }
```

```
        }
        Matriz m1(m,n, mat1);
        Matriz m2(m,n, mat2);
        cout<<endl<<"Matriz m1:"<<endl;
        c2->print(m1);
        cout<<endl<<"Matriz m2:"<<endl;
        c2->print(m2);
        cout<<endl<<"m1+m2:"<<endl;
        c2->print(c2->add(m1, m2));
        cout<<endl<<"m1-m2:"<<endl;
        c2->print(c2->sub(m1, m2));
        cout<<endl<<"m1*m2:"<<endl;
        c2->print(c2->mul(m1, m2));
        cout<<endl<<"m1/m2:"<<endl;
        c2->print(c2->div(m1, m2));

        return 0;
}
```

2. Conclusiones

1. Las clases emplantilladas simplifican enormemente el manejo de múltiples clases, encapsulándolas en un solo archivo.

2. Una plantilla sirve para ahorrarse código, ya que se puede usar la misma con diferentes tipos de variables, o como en este caso, con objetos de diferentes clases.