Reporte de Laboratorio 3

Emmanuel - B51296

27 de septiembre de 2016

Índice

---

1. Introducción

Este laboratorio tuvo como objetivo aprender sobre la creación de clases emplantilladas, así como ampliar los conocimientos de sobrecarga de operadores

1.1. Objetivos

- Aprender acerca de la construcción de clases emplantilladas en C++.
- Aprender sobre sobrecarga de operadores en C++.

2. Código

2.1. Fracción.h

```
#ifndef FRACCION_H
#define FRACCION_H

#include "Calculadora.h"
#include <iostream>
#include "string"

using namespace std;

class Fraccion {
public:
double num;
double den;

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 */
```

2.2. Fracción.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;

}
```

2.3. Polinomio.h

```
#ifndef POLINOMIO_H
#define POLINOMIO_H

//#include "Calculadora.h"
#include <iostream>
#include "string"

using namespace std;

class Polinomio {
public:
int tam;
char var;
double* coef;
```

```
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 */
```

2.4. Polinomio.cpp

```
#include "Polinomio.h"

Polinomio::Polinomio() {
}
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];
}
}
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;
}
}
}
}
```

2.5. Matriz.h

```
#ifndef MATRIZ_H
#define MATRIZ_H

//#include "Calculadora.h"
#include <iostream>
#include "string"

using namespace std;

class Matriz{
public:
int m;
int n;
double **matrix;

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

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 */
```

2.6. Matriz.cpp

```
#include "Matriz.h"
#include <cstdlib>
Matriz::Matriz()
{

}
Matriz::Matriz(int m, int n, double** matrix)
{
this->m =m;
this->n =n;
```

```
this->matrix =matrix;
}
Matriz::~Matriz()
{
}

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;
}
}
```

2.7. Calculadora.h

```
#ifndef CALCULADORA_H
#define CALCULADORA_H

#include <cstdlib>
#include <iostream>
#include "string"
#include "Fraccion.h"
#include "Polinomio.h"
#include "Matriz.h"

template <typename data>
class Calculadora {
public:
Calculadora() {
}
~Calculadora() {
}
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 */
```

2.8. Circulo.cpp

```
#include "Figura.h"
#include "Circulo.h"
```

```
const double PI  =3.14159265358979323846;
///Constructor de la clase circulo.
Circulo::Circulo(){
}
///Destructor de la clase derivada circulo.
Circulo::~Circulo(){
}
///Constructor sobrecargado de la clase derivada circulo.
Circulo::Circulo(string nombre, string color, double r){
this-> nombre = nombre;
this-> color = color;
this-> r=r;

}

///Funcion general para calculo de area de un circulo.
double Circulo::area(){
double a = PI*pow(this-> r, 2);
return a;
}

///Funcioon general para calculo de perimetro de un circulo.
double Circulo::perimetro(){
double p;
p = 2*PI*this->r;
return p;
}

///Desplegador de los datos generales del circulo.
void Circulo::operator~(){
cout << "Nombre: " << this-> nombre <<endl;
cout << "Color: " << this-> color <<endl;
cout << "Longitud del radio " << r <<endl;
cout<< "\n" << endl;

}

///Desplegador del area y el perimetro del circulo.
void Circulo::operator!(){
cout << "Area: " << this-> area() <<endl;
cout << "Perimetro: " << this-> perimetro() <<endl;
cout<< "\n" << endl;

}
```

3. Conclusiones

Gracias a este laboratorio, se consiguió obtener las habilidades necesarias para crear múltiples clases y meterlas en una misma plantilla, simplificando muchos procesos a la hora de realizar programación orientada a objetos.