# Universidad de Costa Rica

# Ciencias de Computación e Informática

# CI-1201 Programación II

# Profesor Adolfo Di Mare

# Cambio al Rep de la calculadora de números racionales

# Gabriel Castañaza B31520

# gcmelmejor@gmail.com

## Introducción

“Con alguna frecuencia es necesario hacer aritmética con números racionales, para lo que conviene contar con una clase números racionales.”

La tarea consiste en modificar una clase C++ cambiándole por completo el Rep. Se reemplazará los campos num y den, que sirven para el denominador y el numerador de la clase “rational” y en su lugar se usará una lista std::list<> de la biblioteca estándar.

## Descripción del problema

El principal problema es que para cambiar el Rep de una clase, hay que tener cuidado, ya que los campos originales son usados continuamente por toda la clase. Por ello el consejo de “no metérsele al Rep” tiene sentido incluso dentro de los métodos de la misma clase, eso permite hacer menos cambios para ajustar de nuevo el Rep.

Para que esto funcione, se recomienda obtener el valor de un campo usando un método especifico para cada campo.

Además se nos pide que hagamos dos versiones de la clase “rational”, una que funcione con la clase calculadora “p2-ta-3.cpp” y otra que funcione con “test\_rational.cpp”. Esto se explicara detalladamente mas adelante.

Y por ultimo documentar la clase calculadora usando Doxygen.

## Cambiar el Rep de la clase “rational”

La clase rational implementa las operaciones aritméticas principales para números racionales.

* [1/3] == [2/6] == ... [9/27] == ...
* [1/3] \* [2/6] / [3/9] - [9/27]

Y para ello utiliza dos campos “m\_den”, “m\_num” que son de tipo emplantillado, para poder usar ya sea int o long, etc.

Lo primero que hay que hacer es cambiar esos dos campos por solo un campo de tipo lista.

private:

std::list<INT> m\_Rep;

Como se puede ver, hay varias cosas a tomar en cuenta. En primer lugar, que el campo debe ser privado, y esto es para, adivine que… “no se le metan al Rep”

Por otro lado, la lista esta emplantillada, y por último el nombre es significativo.

Pero el primer problema que encontraremos es que toda la clase hace referencia a los campos anteriores, así que simplemente no va a compilar. El truco que usaremos es hacer dos métodos uno para obtener el numerador y otro para el denominador. Pero estos métodos deben estar duplicados, una versión constante que será publica y una versión privada que se podrá modificar.

public:

const INT& num() const { return \*(m\_Rep.begin()); }

const INT& den() const {

return \*( ++(m\_Rep.begin()) ) ; }

private:

// Versión modificadora de num().

INT& num() { return \*(m\_Rep.begin()); }

// Versión modificadora de den().

INT& den() {

typename std::list<INT>::iterator it

= m\_Rep.begin();

++it;

return \*it;

}

Se usan varios trucos como incrementar un iterador, para obtener el segundo elemento de la lista.

## Versión Simple

Se nos pidió que en la carpeta “simple” hiciéramos solo los cambios necesarios, para que el programa calculadora compile y se ejecute correctamente.

Para saber que métodos son necesarios modificar, nos fijamos en la clase calculadora y cuando se usan los métodos u operadores en la clase rational

rational<long> r,

acum;

...

std::cout << acum...// operador <<

std::cin >> r; // operador >>

...

case '+': acum.operator+=(r); // operador +=

case '-': acum = operator-(acum,r); // operador -

case '\*': acum \*= r; // operador \*=

case '/': acum = acum / r; // operador /

Luego vamos a la clase “rational” y modificamos cada uno de los operadores de la clase cambiando los campos anteriores a los métodos que creamos.

.m\_num

// y después:

.num()

¿Porque la clase funciona si no hemos actualizado todos los métodos?

Esto es debido a que el compilador de C++ no compila los métodos que no son invocados. Por esa razón funciona sin haber terminado de actualizar la clase completamente.

## Versión Completo

A continuación seguimos actualizando el resto de métodos, usando la opción de buscar y reemplazar.