Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1474

Práctica 1

Tecnología de Programación
Grado en Ingeniería Informática

2026-01-21

Objetivos

- Recordar los conceptos básicos de la programación imperativa en C++.
- Presentar los aspectos básicos de la sintaxis en programación orientada a objetos mediante la implementación de tipos abstractos de datos sencillos.
- Comparar la implementación de un tipo de datos mediante la programación imperativa pura y la programación orientada a objetos.
- Presentar la definición de operadores en C++.

En esta práctica vas a implementar una sencilla calculadora de números racionales, mediante la implementación de un tipo de datos **`Rational`**.

La implementación se hará de dos formas distintas, mediante un **TAD** con las técnicas que ya conoces, y transformando ese TAD a una **clase** de C++, con su correspondiente implementación siguiendo la notación de la Programación Orientada a Objetos.

Como trabajo opcional, se propone la modificación de la implementación de la clase para hacerla más estándar usando la sobrecarga de operadores.

1. TAD en C++

Para implementar esta tarea debes partir de los ficheros que te facilitamos en la carpeta `rational-tad` del material de la práctica.

1.1. Tarea 1

Define e implementa mediante un registro (`struct`) un TAD **`Rational`** que represente un número racional con las operaciones:

- Inicialización con un valor por defecto de 0/1.
- Inicialización a partir de dos enteros (numerador y denominador).
- Lectura de un stream.
- Escritura en un stream.
- Operaciones aritméticas de suma, resta, multiplicación y división entre números racionales.
- Operaciones lógicas de comparación entre números racionales (igual, menor, mayor).

Despues de la inicialización, la lectura y de cualquiera de las operaciones, el número racional resultante debe quedar simplificado.

Utiliza las técnicas que ya conoces (miembros privados, funciones amigas) para limitar el acceso a la información interna de tu TAD.

Deberás implementar las funciones necesarias y con el interfaz adecuado, para que el programa principal que se da con los ficheros correspondientes a esta tarea funcione **sin modificaciones**. Para ello utiliza el esqueleto de los ficheros `rational.{h,cc}` que encontrarás en el directorio `rational-tad` del material de la práctica.

El funcionamiento de tu programa debe ser similar al siguiente (se destaca en negrita lo que teclea el usuario):

```
> main ⏎
? 1/2 + 3/4 ⏎
= 5/4
? 3/7 / 2/3 ⏎
= 9/14
? 1/2 = 2/4 ⏎
= yes
? 2/7 > 5/8 ⏎
= no
? 2/7 = 5/8 ⏎
= no
? 2/7 < 5/8 ⏎
= yes
? . ⏎
>
```

El programa termina cuando se introduce cualquier dato erróneo (por ejemplo un '.'). Puedes detectar esa condición mediante la funcion `cin.fail( )` (ver `main.cc`).

2. Clase en C++

Para implementar esta tarea debes partir de los ficheros que te facilitamos en la carpeta `rational-class` del material de la práctica.

2.1. Tarea 2

Convierte la implementación anterior del tipo **`Rational`**, de un TAD con un registro y funciones que operan sobre él a una **clase** con sus correspondientes métodos. Ten en cuenta que:

- No existen funciones `init(...)`. Casi la totalidad de los lenguajes orientados a objetos permiten la definición de constructores, que hacen las veces de inicializadores. Tienen el mismo nombre que la clase, y se ejecutan automáticamente cuando se crea en memoria el objeto correspondiente.
- Al contrario que en las funciones que operan sobre el TAD (en las que el parámetro que representa al dato del tipo correspondiente se declara explícitamente) en una clase los métodos tienen un parámetro implícito `'this'`, que en C++ es un puntero que apunta al objeto y que permite acceder a sus atributos y métodos. En la mayoría de los casos de uso se puede omitir su uso explícito, aprovechando el concepto de *ámbito de clase*.
- Existen funciones que se pueden convertir en métodos (aquellas en las que el primer parámetro es el objeto) y otras que deben seguir implementándose como funciones externas a la clase. Deberás decidir qué forma de implementación se adapta mejor a cada una, dependiendo de cómo se usan en el programa principal.

Deberás implementar los métodos y funciones necesarios y con el interfaz adecuado para que el programa principal que se da con los ficheros correspondientes a esta tarea funcione **sin modificaciones**. Para ello utiliza el esqueleto de los ficheros `rational.{h,cc}` que encontrarás en el directorio `rational-class` del material de la práctica, y decide cómo debe ser el interfaz de cada método o función para seguir los estándares de C++.

(continúa en la siguiente página...)

EJERCICIO OPCIONAL

3. Sobrecarga de operadores

C++ permite la re–definición de operadores para clases, de forma que el uso de esas clases sea mucho más parecido al de los tipos de datos predefinidos. Esto es especialmente útil para tipos numéricos, como en el caso del tipo `Rational` que acabas de implementar. En lugar de escribir código de este tipo:

```
Rational a,b,c,r;
a.read(cin);
b.read(cin);
c.read(cin);
r = a.add( b.mul(c) );
r.write(cout);
cout << endl;
```

podemos definir los operadores adecuados para poder escribirlo así:

```
Rational a,b,c,r;
cin >> a >> b >> c;
r = a + b*c;
cout << r << endl;
```

Para implementar esta tarea debes partir de los ficheros que te facilitamos en la carpeta `rational-class-op` del material de la práctica.

3.1. Tarea 3

Modifica los métodos y funciones necesarios respecto a la clase `Rational` implementada en la tarea 2 para convertir:

- los métodos que implementan operaciones aritméticas (`add`, `sub`, `mul`, `div`) en operadores infijos (`'+' '-' '*' '/'`). En el fichero `'rational.h'` que se da con el material para esta tarea puedes observar que, para comparar las dos formas de implementarlos, los operadores `'+'` y `'-'` se implementarán como métodos, y los operadores `'*'` y `'/'` como funciones externas a la clase (con sus correspondientes declaraciones `friend`).
- los métodos que implementan operaciones lógicas (`equal`, `lesser_than`, `greater_than`) en operadores infijos (`'==' '<' '>'`).
- los métodos que implementan entrada y salida con *streams* (`read, write`) en operadores infijos (`'>>' '<<'`).

Deberás implementar los métodos y funciones necesarios y con el interfaz adecuado, para que el programa principal que se da con los ficheros correspondientes a esta tarea funcione **sin modificaciones**. Para ello utiliza el esqueleto de los ficheros `rational.{h,cc}` que encontrarás en el directorio `rational-class-op` del material de la práctica, y decide cómo debe ser el interfaz de cada método o función para seguir los estándares de C++.

(continúa en la siguiente página...)

Entrega

Los archivos de código fuente de la práctica deberán estar organizados respetando la estructura y nombres de directorios y ficheros del material entregado para la práctica (XXXXXX es tu NIP, o los NIPs de la pareja de prácticas, ver más adelante):

```
practica1_XXXXXX
     \---- rational-tad
          \---- rational.h
          \---- rational.cc
          \---- main.cc
          \---- Makefile
     \---- rational-class
          \---- rational.h
          \---- rational.cc
          \---- main.cc
          \---- Makefile
     \---- rational-class-op
          \---- rational.h
          \---- rational.cc
          \---- main.cc
          \---- Makefile
```

Cada subdirectorio no deberá incluir otros archivos de código fuente, y los nombres de los archivos serán los indicados en la práctica. Recuerda que no debes modificar los archivos correspondientes al programa principal (`main.cc`) en ninguna de las tareas.

El programa en C++ deberá ser compilable y ejecutable con los archivos que has entregado en cada subcarpeta mediante los siguientes comandos:

```
> make
...
> ./main
```

No se deben utilizar paquetes, bibliotecas, ni ninguna infraestructura adicional fuera de las bibliotecas estándar del propio lenguaje.

En caso de no compilar siguiendo estas instrucciones, **la nota de la evaluación de la práctica será un `0`**.

Todos los archivos de código fuente (y directorios, si es el caso) solicitados en este guión deberán ser comprimidos en un único archivo ZIP con el siguiente nombre:

- **`practica1_<nip1>_<nip2>.zip`** (donde <nip1> y <nip2> son los NIPs de los estudiantes involucrados) si el trabajo ha sido realizado en pareja.
  En este caso sólo uno de los dos estudiantes deberá hacer la entrega en Moodle.
- **`practica1_<nip>.zip`** (donde <nip> es el NIP del estudiante involucrado) si el trabajo ha sido realizado de forma individual.

El archivo comprimido a entregar no debe contener ningún fichero aparte de los fuentes (y en su caso el fichero `Makefile`) que te pedimos: ningún fichero ejecutable (`*.exe`) u objeto (`*.o`), clases compiladas de Java (`*.class`), ficheros de interfaz de Haskell (`*.hi`), archivos de configuración del entorno de desarrollo (`.vscode`) ni ningún otro fichero adicional.

La entrega del archivo ZIP deberá hacerse en la tarea correspondiente preparada en el curso Moodle de la asignatura y antes de la fecha límite fijada para ello.