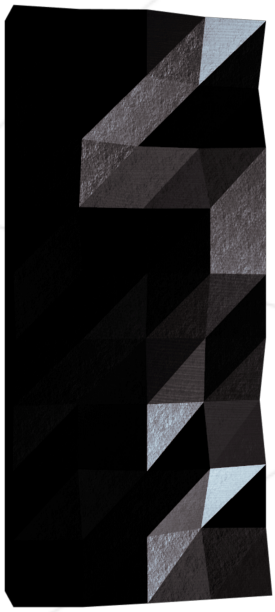

C++ - Módulo 02

Polimorfismo ad-hoc, overloads y clases canónicas

Resumen: Este documento contiene el módulo 02 de los módulos C++ de 42.

Índice general

Capítulo I

Reglas Generales

- La declaración de una función en un header (excepto para los templates) o la inclusión de un header no protegido conllevará un 0 en el ejercicio.
- Salvo que se indique lo contrario, cualquier salida se mostrará en stdout y terminará con un newline.
- Los nombres de ficheros impuestos deben seguirse escrupulosamente, así como los nombres de clase, de función y de método.
- Recordatorio : ahora está codificando en `C++`, no en `C`. Por eso :
  - Las funciones siguientes están **PROHIBIDAS**, y su uso conllevará un 0: `*alloc`, `*printf` et `free`
  - Puede utilizar prácticamente toda la librería estándar. NO OBSTANTE, sería más inteligente intentar usar la versión para C++ que a lo que ya está acostumbrado en C, para no basarse en lo que ya ha asimilado. Y no está autorizado a utilizar la STL hasta que le llegue el momento de trabajar con ella (módulo 08). Eso significa que hasta entonces no se puede utilizar Vector/List/Map/etc... ni nada similar que requiera un include <algorithm>.
- El uso de una función o de una mecánica explícitamente prohibida será sancionado con un `0`
- Tenga también en cuenta que, a menos que se autorice de manera expresa, las palabras clave `using namespace` y `friend` están prohibidas. Su uso será castigado con un `0`.
- Los ficheros asociados a una clase se llamarán siempre `ClassName.cpp` y `ClassName.hpp`, a menos que se indique otra cosa.
- Tiene que leer los ejemplos en detalle. Pueden contener prerrequisitos no indicados en las instrucciones.
- No está permitido el uso de librerías externas, de las que forman parte `C++11`, `Boost`, ni ninguna de las herramientas que ese amigo suyo que es un figura le ha recomendado.
- Probablemente tenga que entregar muchos ficheros de clase, lo que le va a parecer repetitivo hasta que aprenda a hacer un script con su editor de código favorito.

- Lea cada ejercicio en su totalidad antes de empezar a resolverlo.
- El compilador es `clang++`
- Se compilará su código con los flags `-Wall -Wextra -Werror`
- Se debe poder incluir cada include con independencia de los demás include. Por lo tanto, un include debe incluir todas sus dependencias.
- No está obligado a respetar ninguna norma en `C++`. Puede utilizar el estilo que prefiera. Ahora bien, un código ilegible es un código que no se puede calificar.
- Importante: no va a ser calificado por un programa (a menos que el enunciado especifique lo contrario). Eso quiere decir que dispone cierto grado de libertad en el método que elija para resolver sus ejercicios.
- Tenga cuidado con las obligaciones, y no sea zángano; podría dejar escapar mucho de lo que los ejercicios le ofrecen.
- Si tiene ficheros adicionales, no es un problema. Puede decidir separar el código de lo que se le pide en varios ficheros, siempre que no haya moulinette.
- Aun cuando un enunciado sea corto, merece la pena dedicarle algo de tiempo, para asegurarse de que comprende bien lo que se espera de usted, y de que lo ha hecho de la mejor manera posible.

Capítulo II

Reglas de la parte extra

- A partir de ahora, cada clase que escriba DEBERÁ ser canónica. Tendrá al menos un constructor por defecto, un constructor de copia, un overload de operadores de asignación y un destructor.

Capítulo III

Ejercicio 00: Mi primera clase canónica

	Ejercicio : 00
Ejercicio 00: Mi primera clase canónica	
Directorio de entrega : $ex00/$	
Ficheros a entregar : `Fixed.hpp y Fixed.cpp`	
Funciones prohibidas : `Ninguna`	

Ya conoce los int y los floats. Qué majo.

Léase este artículo de 3 páginas (en inglés, ndlr): (1, 2, 3) y se dará cuenta de que no es así. Vamos, léalo.

Hasta ahora, los números que ha utilizado en sus programas eran enteros o decimales, o alguna de sus variantes (`short`, `char`, `long`, `double`, etc.). Según lo que acaba de leer, sería razonable pensar que los números enteros y los números de coma flotante tengan características opuestas.

Pero hoy, todo eso va a cambiar. Va a descubrir un nuevo tipo de número genial: ¡los números de coma fija! Ausentes en la mayoría de los lenguajes tipados, los números de coma fija ofrecen un valioso equilibrio entre prestaciones, exactitud, alcance y precisión. Esta es la razón por la que su uso está tan extendido en programas gráficos, de audio y científicos, solo por citar algunos.

Como el `C++` no dispone de números de coma fija, se los vamos a añadir. Para empezar con buen pie, le recomiendo que lea este artículo de `Berkeley`. Si a ellos les vale, a nosotros también. Si no sabe lo que es `Berkeley`, lea esta sección de su página de Wikipedia.

Escriba una clase canónica que represente los números de coma fija:

- Miembros privados:
  - Un int para almacenar el valor de coma fija.
  - Una `static constant int` para almacenar el número de bits fraccionarios. Esta constante siempre valdrá `8`.
- Miembros públicos:
  - Un constructor por defecto que inicialice el valor de coma fija a `0`.
  - Un destructor.
  - Un constructor de copia.
  - Un overload de operador de asignación.
  - Una función miembro `int getRawBits(void) const;` que devuelva el valor bruto del número de coma fija.
  - Una función miembro `void setRawBits(int const raw);` que haga el set del valor del número de coma fija.

Este código:

```
#include <iostream>

int		main(void) {

	Fixed a;
	Fixed b(a);
	Fixed c;

	c = b;

	std::cout << a.getRawBits() << std::endl;
	std::cout << b.getRawBits() << std::endl;
	std::cout << c.getRawBits() << std::endl;

	return 0;
}
```

Debería devolver algo parecido a esto:

```
$> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
$> ./a.out
Default constructor called
Copy constructor called
Assignation operator called // <-- Puede que no esté esta línea
getRawBits member function called
Default constructor called
Assignation operator called
getRawBits member function called
getRawBits member function called
0
getRawBits member function called
0
getRawBits member function called
0
Destructor called
```

```
Destructor called
Destructor called
$>
```

Capítulo IV

Ejercicio 01: Hacia una clase más útil

	Ejercicio : 01
Ejercicio 01: Hacia una clase más útil	
Directorio de entrega : $ex01/$	
Ficheros a entregar : `Fixed.hpp y Fixed.cpp`	
Funciones autorizadas : `roundf (from <cmath>)`	

Bueno, el `ex00` ha sido un buen comienzo, pero nuestra clase realmente no sirve para nada, dado que solo puede representar el valor `0.0`. Por lo tanto, añada a su clase los siguientes constructores y funciones miembro públicos:

- Un constructor que reciba un const int como parámetro y lo convierta en su valor de coma fija(8) correspondiente. Se debe inicializar la parte fraccionaria como en el ex00
- Un constructor que reciba un const float como parámetro y lo convierta en su valor de coma fija(8) correspondiente. Se debe inicializar la parte fraccionaria como en el ex00
- Una función miembro `float toFloat(void) const;` que convierta el valor de coma fija en un valor de coma flotante.
- Una función miembro `int toInt(void) const;` que convierta el valor de coma fija en un valor entero.

También añadirá el siguiente overload a su header y a su archivo fuente:

- Overload de << que inserta una representación (en coma flotante) de su número de coma fija en el output solicitado.

Puede utilizar el siguiente código:
Debería devolver algo parecido a esto:

```cpp
#include <iostream>

int     main(void) {

    Fixed       a;
    Fixed const b(10);
    Fixed const c(42.42f);
    Fixed const d(b);

    a = Fixed(1234.4321f);

    std::cout << "a is " << a << std::endl;
    std::cout << "b is " << b << std::endl;
    std::cout << "c is " << c << std::endl;
    std::cout << "d is " << d << std::endl;

    std::cout << "a is " << a.toInt() << " as integer" << std::endl;
    std::cout << "b is " << b.toInt() << " as integer" << std::endl;
    std::cout << "c is " << c.toInt() << " as integer" << std::endl;
    std::cout << "d is " << d.toInt() << " as integer" << std::endl;

    return 0;
}
```

```
$> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
$> ./a.out
Default constructor called
Int constructor called
Float constructor called
Copy constructor called
Assignation operator called
Float constructor called
Assignation operator called
Destructor called
a is 1234.43
b is 10
c is 42.4219
d is 10
a is 1234 as integer
b is 10 as integer
c is 42 as integer
d is 10 as integer
Destructor called
Destructor called
Destructor called
Destructor called
$>
```

Capítulo V

Ejercicio 02: Now we're talking

	Ejercicio : 02
Ejercicio 02: Now we're talking	
Directorio de entrega : $ex02/$	
Ficheros a entregar : `Fixed.hpp y Fixed.cpp`	
Funciones autorizadas : `roundf (from <cmath>)`	

`Este ejercicio no ofrece ningún punto, pero sigue siendo útil. Puede hacerlo o no.`

Nos vamos acercando. Añada a su clase las funciones miembro públicas y overloads siguientes:

- Seis operadores de comparación: `>`, `<`, `>=`, `<=`, `==` y `!=`.
- Cuatro operadores aritméticos: `+`, `-`, `*` y `/`.
- Los operadores de preincremento, postincremento, predecremento y postdecremento que incrementarán y decrementarán el valor del número de coma fija con el valor representable ϵ más pequeño posible de tal modo que $1 + \epsilon > 1$.

Añada a su clase los overloads de funciones públicas no miembro:

- La función no miembro `min` que recibe dos referencias a números de coma fija y devuelve una referencia al más pequeño y un overload que recibe dos referencias a dos números de coma fija constantes y devuelve una referencia al valor constante más pequeño.

- La función no miembro `max` que recibe dos referencias a números de coma fija y devuelve una referencia al más grande y un overload que recibe dos referencias a dos números de coma fija constantes y devuelve una referencia al valor constante más grande.

Es su responsabilidad probar las distintas características de su clase, pero este fragmento de código:

```
#include <iostream>

int     main(void) {

    Fixed           a;
    Fixed const     b(Fixed(5.05f) * Fixed(2));

    std::cout << a << std::endl;
    std::cout << ++a << std::endl;
    std::cout << a << std::endl;
    std::cout << a++ << std::endl;
    std::cout << a << std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max(a, b) << std::endl;

    return 0;
}
```

debería mostrar algo parecido a esto (hemos eliminado los ctors/dtors logs):

```
$> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```

Capítulo VI

Ejercicio 03: Expresiones

	Ejercicio : 03
Ejercicio 03: Expresiones	
Directorio de entrega : $ex03/$	
Ficheros a entregar : `Fixed.hpp y Fixed.cpp, cualquier archivo que necesite y un Makefile`	
Funciones autorizadas : `roundf (from <cmath>)`	

Este ejercicio no ofrece ningún punto, pero sigue siendo útil. Puede hacerlo o no.

Escriba un programa que se llame `eval_expr` y evalúe expresiones aritméticas sencillas con números de coma fija.

Procure utilizar strings, istreams, ostreams, istringstreams y ostringstreams todo lo que pueda. Así ahorrará mucho tiempo.

Por ejemplo:

```
$> clang++ -Wall -Wextra -Werror -o eval_expr Fixed.class.cpp {your files}
$> ./eval_expr "(18.18 + 3.03) * 2"
42.4219
$>
```