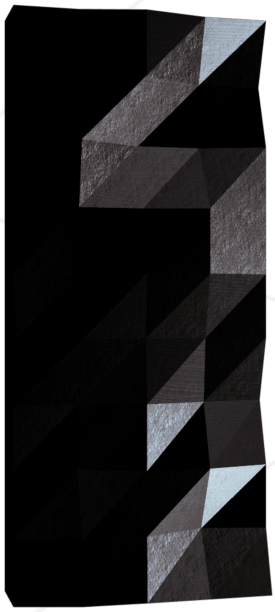

C++ Pool - d02

Ad-hoc polymorphism, operators overload and canonical classes

Staff 42 bocal@staff.42.fr

Abstract: This document contains the subject for day 02 of 42's C++ pool.

Contents

Chapter I

General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in `C++` now, not in `C` anymore. Therefore:
  - The following functions are FORBIDDEN, and their use will be punished by a `-42`, no questions asked: `*alloc`, `*printf` and `free`.
  - You are allowed to use basically everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until d08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.
- Actually, the use of any explicitly forbidden function or mechanic will be punished by a `-42`, no questions asked.
- Also note that unless otherwise stated, the `C++` keywords `"using namespace"` and `"friend"` are forbidden. Their use will be punished by a `-42`, no questions asked.
- Files associated with a class will always be `ClassName.hpp` and `ClassName.cpp`, unless specified otherwise.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand `C++` enough.

- Since you are allowed to use the `C++` tools you learned about since the beginning of the pool, you are not allowed to use any external library. And before you ask, that also means no `C++11` and derivates, nor `Boost` or anything your awesomely skilled friend told you `C++` can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.

- Read each exercise FULLY before starting it ! Really, do it.

- The compiler to use is `clang++`.

- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror`.

- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.

- The subject can be modified up to 4h before the final turn-in time.

- In case you're wondering, no coding style is enforced during the `C++` pool. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.

- Important stuff now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and DO NOT be lazy, you would miss a LOT of what they have to offer !

- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the day is not graded by a program.

- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.

- By Odin, by Thor ! Use your brain !!!

Chapter II

Day-specific rules

- From now on, each class you write MUST be in canonical form : At least one default constructor, a copy contructor, an assignation operator overload and a destructor.

Chapter III

Foreword

How to tell the difference between metal genres ? Easy:

Power Metal: The protagonist arrives riding a white unicorn, escapes from the dragon, saves the princess and makes love to her in an enchanted forest.

Thrash Metal: The protagonist arrives, fights the dragon with his bare hands, saves the princess and fucks her.

Heavy Metal: The protagonist arrives on a Harley Davidson heavily customized, kills the dragon, drinks a few beers with the princess and gets awesome sex from her.

Folk Metal: The protagonist arrives with some friends playing accordions, violins, flutes and many more weird instruments, the dragon falls asleep (because of all the dancing). Then all leave, without the princess.

Viking Metal: The protagonist arrives in a ship, kills the dragon with his mighty axe, skins the dragon and eats it, rapes the princess to death, steals her belongings and burns the castle before leaving.

Death Metal: The protagonist arrives, kills the dragon, fucks the princess and kills her, then leaves.

Black Metal: The protagonist IS the dragon, dwells in the heart of the night within a castle full of hellhounds and eternal flames. He kills the sassy knight, fucks the noble steed and sacrifices the princess to Satan.

Gore Metal: The protagonist arrives, kills the dragon and spreads his guts in front of the castle, fucks the princess and kills her. Then he fucks the dead body again, slashes her belly and eats her guts. Then he fucks the carcass for the third time, burns the corpse and fucks it for the last time.

Doom Metal: The protagonist arrives under heavy rain, sees the size of the dragon and thinks he could never beat him, then he gets depressed and commits suicide. The dragon eats his body and the princess as dessert. That's the end of a sad story.

Progressive Metal: The protagonist arrives with a guitar and plays a solo of 26 minutes. The dragon kills himself out of boredom. The protagonist arrives to the princess' bedroom, plays another solo with all the techniques and tunes he learned in the last year of conservatory. The princess escapes looking for the Heavy Metal protagonist.

Glam Metal: The protagonist arrives, the dragon laughs at the guy's appearance and lets him enter. He steals the princess' make up and tries to paint the castle in a beautiful pink colour.

Nu Metal: The protagonist arrives in a run down Honda Civic and attempts to fight the dragon, but burns to death when his moronic baggy clothes catch fire. The princess escapes by herself.

Grindcore: The protagonist arrives, screams something completely undecipherable for about sixteen seconds then leaves... The princess and the dragon are not sure about what happened.

Gothic Metal: The princess in a velvet costume starts singing soprano. The protagonist completes the duet when he shows up, they sing while the dragon plays the flute. Suddenly the dragon swallows up the pipe and accidentally scorches the beauty and the protagonist, and then he suffocates to death. All their souls are damned in hell for eternity.

Industrial Metal: The protagonist arrives naked, wearing only a gas mask, makes an obscene gestures towards dragon, and gets escorted out of fairy tale land by security guards.

Speed Metal: Suddenly there is a short solo, the dragon is confused; someone's screaming weird stuff; the princess realizes she's been deflowered; the dragon and the princess are still looking for the one who caused this.

Symphonic Metal: The princess enters the lair of the dragon with a full choir and orchestra. Together they sing notes so powerfull that they make the dragon go deaf. The dragon tries to burn them all to death, but the princess and her choir sing so loudly and unified, that it creates a protective bubble which only grows in strength from the lead guitarist's solo later in the song. The flames bounce back killing the dragon. The princess then saves the protagonist before leaving the scene while everybody sing a mind blowing finale.

Chapter IV

Exercise 00: My First Canonical Class

	Exercise 00
Exercise 00: My First Canonical Class	
Turn-in directory : *ex*00/	
Files to turn in : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc. Pick one, I don't care unless it's stupid.`	
Forbidden functions : `None`	
Remarks : `n/a`	

You know integers and you also know floating point numbers. How cute.

Please read this 3-page article (1, 2, 3) to discover that you don't. Go on, read it.

Until today, any numbers you used in your programs were basically integers or floating points numbers, or any of their variants (`short`, `char`, `long`, `double`, etc). From your previous reading, it's safe to assume that integers and floating point numbers have opposite caracteristics.

But today, this will change. You are going to discover a new and awesome number type : fixed points numbers ! Always missing from most languages scalar types, fixed points numbers offer a valuable balance between performances, accuracy, range and precision that explains why these numbers are widely used in graphics, sound or scientific programming to name a few.

As `C++` lacks fixed point numbers, you're going to add them yourself today. I'd recommend this article from `Berkeley` as a start. If it's good for them, it's good for you. If you have no idea what `Berkeley` is, read this section of their wikipedia page.

Write a canonical class to represent fixed point numbers :

- Private members :
  - An integer to store the fixed point value.
  - A `static constant` integer to store the number of fractional bits. This constant will always be the litteral `8`.
- Public members :
  - A default constructor that initializes the fixed point value to `0`.
  - A destructor.
  - A copy constructor.
  - An assignation operator overload.
  - A member function `int getRawBits( void ) const;` that returns the raw value of the fixed point value.
  - A member function `void setRawBits( int const raw );` that sets the raw value of the fixed point value.

The code :

```
#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;
}
```

Should output something like :

```
$> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
$> ./a.out
Default constructor called
Copy constructor called
Assignation operator called   // <-- This line may be missing depending of your implementation
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
$>
```

Chapter V

Exercise 01: Towards a more useful fixed point class

	Exercise 01
Exercise 01: Towards a more useful fixed point class	
Turn-in directory : $ex01/$	
Files to turn in : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc. Pick one, I don't care unless it's stupid.`	
Allowed functions : `roundf (from <cmath>)`	
Remarks : `n/a`	

Ok, `ex00` was a good start, but our class is still pretty useless, being only able to represent the fixed point value `0.0`. Add the following public constructors and public member functions to your class :

- A constructor that takes a constant integer as a parameter and that converts it to the correspondant fixed(8) point value. The fractional bits value is initialized like in `ex00`.
- A constructor that takes a constant floating point as a parameter and that converts it to the correspondant fixed(8) point value. The fractional bits value is initialized like in `ex00`.
- A member function `float toFloat( void ) const;` that converts the fixed point value to a floating point value.
- A member function `int toInt( void ) const;` that converts the fixed point value to an integer value.

You will also add the following function overload to your header (declaration) and source (definition) files :

- An overload to the « operator that inserts a floating point representation of the fixed point value into the parameter output stream.

The code :

```
#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;
}
```

Should output something like :

```
$> 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
$>
```

Chapter VI

Exercise 02: Now we're talking

	Exercise 02
Exercise 02: Now we're talking	
Turn-in directory : $ex02/$	
Files to turn in : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc. Pick one, I don't care unless it's stupid.`	
Allowed functions : `roundf (from <cmath>)`	
Remarks : `n/a`	

We're getting closer. Add the following public member operator overloads to your class :

- Six comparison operators : `>`, `<`, `>=`, `<=`, `==` and `!=`.
- Four arithmetic operators : `+`, `-`, `*`, and `/`.
- The pre-increment, post-increment, pre-decrement and post-decrement operators, that will increment or decrement the fixed point value from the smallest representable ϵ such as $1 + \epsilon > 1$.

Add the following public non member functions overloads to your class :

- The non-member function `min` that takes references on two fixed point values and returns a reference to the smallest value, and an overload that takes references on two constant fixed point values and returns a reference to the smallest constant value.
- The non-member function `max` that takes references on two fixed point values and returns a reference to the biggest value, and an overload that takes references on two constant fixed point values and returns a reference to the biggest constant value.

It's up to you to test every feature of your class, but the short code :

```
#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;
}
```

Should output something like (I deleted the ctors/dtor 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
$>
```

Chapter VII

Exercise 03: Food for thought

	Exercise 03
Exercise 03: Food for thought	
Turn-in directory : *ex*03/	
Files to turn in : `n/a`	
Forbidden functions : `None`	
Remarks : `n/a`	

Choose one of you neighbors, and ask her/him :

"What do you think of using a namespace instead of a class to represent fixed point values ? I mean, with a scoped typedef on int, we could write scoped functions and save the cost of instanciations, but at the same time, we'd loose some handy syntax. What's your point of view ?"

This exercice won't give you points. It's just a little break in the day to take time to look back at what you learned during the past three days.

If you want to skip this, fine. I don't care. You will probably end up being a web developper anyway `>.>` !

Chapter VIII

Exercise 04: Fixed point expressions

	Exercise 04
Exercise 04: Fixed point expressions	
Turn-in directory : *ex*04/	
Files to turn in : `Fixed.class.hpp and Fixed.class.cpp, plus any additionnal files you need, and a Makefile. The naming convention is up to you.`	
Allowed functions : `roundf (from <cmath>)`	
Remarks : `n/a`	

Write a program named `eval_expr` that evaluates simple arithmetic expressions as fixed point values.

`Please use strings, istreams, ostreams, istringstreams and ostringstreams as much as you can. These will save you a LOT of time.`

For instance :

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