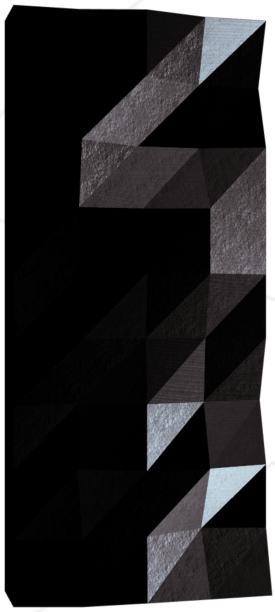

C++ Pool - rush00

ft_retro

Staff 42 bocal@staff.42.fr

Abstract: This document contains the subject for rush 00 of 42's C++ pool, which if done correctly will help you lose all your friends and/or the game.

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

- Classes will have to be in Coplien's form, without us asking. It's a pre-requisite to have a positive grade. Be thorough.
- Your program must be compiled with a proper Makefile, and the executable must be named `ft_retro`.

Chapter III

Foreword

Here's what Wikipedia has to say on `Battlestar Galactica` :

`Battlestar Galactica is an American science fiction franchise created by Glen A. Larson. The franchise began with the original television series in 1978 and was later followed by a short-run sequel series (Galactica 1980), a line of book adaptations, original novels, comic books, a board game, and video games. A re-imagined version of Battlestar Galactica aired as a two-part, three-hour miniseries developed by Ronald D. Moore and David Eick in 2003. That miniseries led to a weekly television series, which later aired up until 2009. A prequel series, Caprica, aired in 2010. A two-hour pilot for a second spin-off prequel series, Blood & Chrome, aired in 2013 though this did not lead to a series as originally planned.`

`All Battlestar Galactica productions share the premise that in a distant part of the universe, a human civilization has extended to a group of planets known as the Twelve Colonies, to which they have migrated from their ancestral homeworld of Kobol. The Twelve Colonies have been engaged in a lengthy war with a cybernetic race known as the Cylons, whose goal is the extermination of the human race. The Cylons offer peace to the humans, which proves to be a ruse. With the aid of a human named Baltar, the Cylons carry out a massive attack on the Twelve Colonies and on the Colonial Fleet of starships that protect them. These attacks devastate the Colonial Fleet, lay waste to the Colonies, and virtually destroy their populations. Scattered survivors flee into outer space aboard a ragtag array of available spaceships. Of the entire Colonial battle fleet, only the Battlestar Galactica, a gigantic battleship and spacecraft carrier, appears to have survived the Cylon attack. Under the leadership of Commander Adama, the Galactica and the pilots of "Viper fighters" lead a fugitive fleet of survivors in search of the fabled thirteenth colony known as Earth.`

There is very little chance of you having a good grade on this project if you have not watched `Battlestar Galactica` in its entirety.

Chapter IV

Subject

First off, remember one thing: This is a C++ project. Therefore, if you try to turn in a "C+" project, and by that we mean any kind of project that's mostly C with just one or two classes thrown around for show, std::cout instead of printf, etc ... (You know what we mean), then you will not be graded at all. In other words, if you are not using the object-oriented properties of the language, you're not really coding in C++.

The goal of this project is to implement a simplistic shoot-em-up-style game in your terminal. For those of you who don't know what that kind of game is (shame on you, by the way), have a look at `Gradius`, `R-Type`, etc...

You will use a 'screen' made up of a grid of 'squares', that you can equate to the characters on your terminal, so that the entities of your game are each represented by a character on screen.

Here are the basic requirements :

- Single-player
- Display using the `ncurses` library
- Horizontal or vertical scrolling (The screen area moves through the world, very much like in R-Type for example)
- Random enemies
- The player can shoot at enemies
- Basic collision handling (If an enemy touches you, you die)
- Game entities occupy one 'square' of the map only.

- Frame-based timing

It may seem a little daunting, but the basic requirements can be fulfilled by a game even simpler and way uglier than the already pretty simple `Space Invaders`. So it's not actually that hard.

There will be plenty of occasions to use the various facilities of C++. For example, representing the various entities of your game (Player ship, enemies, missiles, etc ...) as subclasses of a GameEntity class ? Maybe even an interface ? At any rate, a very large part of the grading will regard intelligent use of objects, inheritance, and so on.

For those of you who have never made a video game, here's how a very basic game should run:

- Acquire input (Player controls, network, etc ...)
- Update game entities
- Render display
- Repeat until game ends !

Of course, the "Update" phase includes a lot of things, including handling the spawning of enemies, making actions required by player input, moving entities according to their current vector and the time since the last update ... But those are the basics. When you get down to it, a basic video game is a really simple program. It should not be too hard to do, should it ?

When you're pretty sure you've done everything right, you are very welcome to improve on the basic requirements and make an even better game.

Here are some of the features we consider "essential" and that will have to be done before other bonuses are counted:

- Displaying score, time, number of lives, etc... on screen
- Clock-based timing (Use whichever system facility or library you like)
- Entities that can occupy multiple squares
- Enemies can also shoot
- Scenery (Collidable objects or simple background)

And after that, if you still want something to do, here are some ideas of cool bonuses, but keep in mind that we'll consider them useless if you haven't done the other stuff before:

- Large and hard-to-beat boss enemies
- Enemies have a scripted behaviour
- Multiplayer, either on the same keyboard or through the network if you're feeling cocky
- Scripted game worlds, with pre-determined batches of enemies

Keep in mind that, as usual, bonuses will not be graded if the basics are not completed, and a bonus that crashes your program could bring your whole grade down...