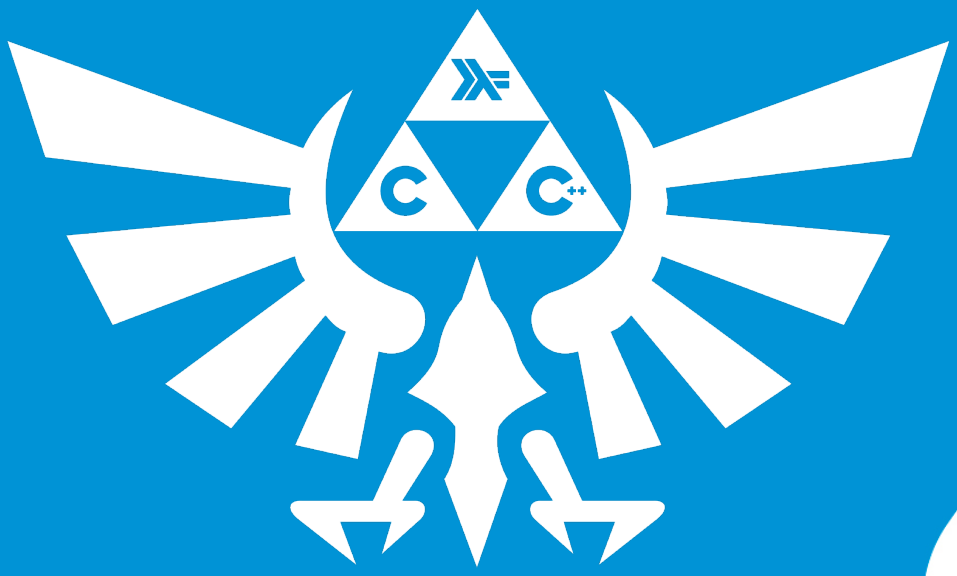
{EPITECH}

DAY 08

THE TRADE FEDERATION



DAY 08

All your exercises will be compiled with `g++` **and the** `-std=c++20 -Wall -Wextra -Werror` **flags**, unless specified otherwise.

All output goes to the standard output, and must be ended by a newline, unless specified otherwise.

None of your files must contain a `main` function, unless specified otherwise. We will use our own `main` functions to compile and test your code. It will include your header files.

For each exercise, the files must be turned-in in a separate directory called **exXX** where XX is the exercise number (for instance `ex01`), unless specified otherwise.

Read the examples CAREFULLY. They might require things that weren't mentioned in the subject...

The `*alloc`, `free`, `*printf`, `open` and `fopen` functions, as well as the `using namespace` keyword, are forbidden in C++. By the way, `friend` is forbidden too, as well as any library except the standard one.

Unit Tests

It is highly recommended to test your functions as you implement them. It is common practice to create and use what are called **unit tests**.

From now on, we expect you to write unit tests for your functions (when possible). To do so, please follow the instructions in the **"How to write Unit Tests"** document on the intranet, available here.

For them to be executed and evaluated, put a `Makefile` at the root of your directory with the `tests_run` rule as mentionned in the documentation linked above.

Exercise 0 - Droids

Turn in : `Droid.hpp`, `Droid.cpp`

Hey you ! Yes… you, over there. From now on, you are a lead designer. What for ? Well, you are now chief engineer designer for my upcoming **Droid army** ! Why you ? Just because you were there. Stop babbling now, and get to work.

Start by creating a cheap Droid with the following specifications :

- ✓ The `Droid` takes as a parameter its serial number, which is an `std::string`. The `Droid` can be constructed without this serial number. In this case, the serial number is an empty string.
- ✓ The `Droid` has a copy constructor for replication, as well as an assignment operator for replacement. This is the easiest solution to damaged `Droids`.
- ✓ The `Droid` also has the following properties :

  `Id`, the `Droid`'s serial number, stored as an `std::string`
  `Energy`, the remaining energy before the `Droid`'s batteries need to be changed, stored as a `size_t`
  `Attack`, the `Droid`'s attack power, stored as a `const size_t`
  `Toughness`, the `Droid`'s resistance, stored as a `const size_t`
  `Status`, the `Droid`'s current status, stored as an `std::string *`

  Upon construction, `Energy`, `Attack`, `Toughness` and `Status` are respectively set to 50, 25, 15 and "Standing by".

  Each of these attributes is private. They therefore have a getter, the form of which is `get[Property]`, and a setter, the form of which is `set[Property]`.
  `const` values have no setter, obviously.
- ✓ The `Droid` is in charge of its `Status` and takes ownership of it. The `Droid` is in charge of its destruction (meaning no memory leaks).
- ✓ It is necessary to know whether two `Droids` are identical or not, thanks to the == and '!=' operators. Be careful : we don't care whether we are comparing the **same** `Droid`.
  Two `Droids` are considered identical if they have the same characteristics.
- ✓ Overload the << operator to reload the `Droid`.
  A `Droid` can't have more than 100 nor less than 0 `Energy`. It substracts the value it requires to reload its batteries from the other operand. It must be possible to chain calls. An example is shown below.