#### SCALE FOR PROJECT CPP MODULE 04

You should evaluate 1 student in this team

#### Introduction

Please comply with the following rules:

- Remain polite, courteous, respectful and constructive throughout the evaluation process. The well-being of the community depends on it.
- Identify with the student or group whose work is evaluated the possible dysfunctions in their project. Take the time to discuss and debate the problems that may have been identified.
- You must consider that there might be some differences in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade them as honestly as possible. The pedagogy is useful only and only if the peer-evaluation is done seriously.

#### Guidelines

- Only grade the work that was turned in the Git repository of the evaluated student or group.
- Double-check that the Git repository belongs to the student(s). Ensure that the project is the one expected. Also, check that 'git clone' is used in an empty folder.
- Check carefully that no malicious aliases was used to fool you and make you evaluate something that is not the content of the official repository.
- To avoid any surprises and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).
- If you have not completed the assignment you are going to evaluate, you have to read the entire subject prior to starting the evaluation process.
- Use the available flags to report an empty repository, a non-functioning program, a Norm error, cheating, and so forth. In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except for cheating, student are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated in the future.
- You should never have to edit any file except the configuration file if it exists. If you want to edit a file, take the time to explicit the reasons with the evaluated student and make sure both of you are okay with this.
- You must also verify the absence of memory leaks. Any memory allocated on the heap must be properly freed before the end of execution. You are allowed to use any of the different tools available on the computer, such as leaks, valgrind, or e\_fence. In case of memory leaks, tick the appropriate flag.

# **Attachments**

subject.pdf

# **Preliminary tests**

If cheating is suspected, the evaluation stops here. Use the "Cheat" flag to report it. Take this decision calmly, wisely, and please, use this button with caution.

## Prerequisites

The code must compile with c++ and the flags -Wall -Wextra -Werror Don't forget this project has to follow the C++98 standard. Thus, C++11 (and later) functions or containers are NOT expected.

Any of these means you must not grade the exercise in question:

- A function is implemented in a header file (except for template functions).
- A Makefile compiles without the required flags and/or another compiler than c++

Any of these means that you must flag the project with "Forbidden Function":

- Use of a "C" function (\*alloc, \*printf, free).
- Use of a function not allowed in the exercise guidelines.
- Use of "using namespace <ns\_name>" or the "friend" keyword.

  Use of an external library, or features from versions other than C++98.



# Exercise 00: Polymorphism

As usual, there has to be enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

### First check

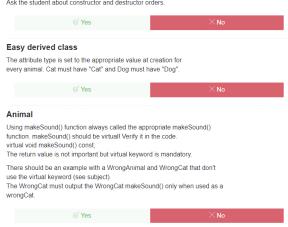
There is an Animal class that has one attribute: One string called type. You must be able to instantiate and use this class



### Inheritance

They are at least two classes that inherit from Animal: Cat and Dog. The constructor and destructor outputs must be clear.

Ask the student about constructor and destructor orders



# Exercise 01: I do not want to set the world on fire

As usual, there has to be enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

#### Concrete Animal

There is a new class called Brain.

Cat and Dog have the required private Brain attribute.

The Brain attribute should not be inside the Animal class.

The Brain class has specific outputs upon creation and deletion



# **Destruction chaining**

The destructors in Animal must be virtual.

Ask an explanation of what will happen without the virtual keyword over Animal destructor

Test it.



# Assignment and copy

The copy and assignment behaviors of the Cat and Dog are like the subject requires.

Deep copy means you need to create a new Brain for the Cat or Dog Check that the canonical form is really implemented (i.e. no empty copy assignment operators and so forth). Nothing should be public for no reason. Moreover, this code is very simple so it needs to be clean!



# Exercise 02: Abstract class

As usual, there has to be enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

### Abstract class

There is an Animal class exactly like the one in the subject.

The Animal::makeSound is a pure virtual function

It should look like : virtual void makeSound() const = 0; The "= 0" part is mandatory.

You should not be able to instantiate an Animal.

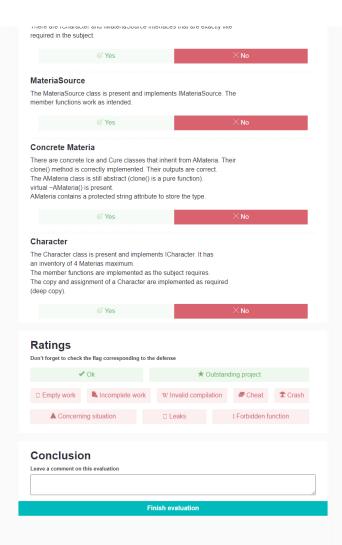
Animal test; //should give you a compile error about the class being abstract



# Exercise 03: Interface and recap

As usual, there has to be enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise

### Interfaces



Give this repository a star. 🏫