SCALE FOR PROJECT CPP MODULE 05

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.

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 " or the "friend" keyword.
- Use of an external library, or features from versions other than C++98.

☑ Yes ☒ No

Ex00: Mommy, when I grow up, I want to be a bureaucrat!

As usual, there has to be the main function that contains 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.

Bureaucrat

There is a Makefile that compiles using the appropriate flags.
There is a Bureaucrat class. It has a constant name.
It has a grade that ranges from 1 (highest) to 150 (lowest).
Exceptions are thrown when trying to create a Bureaucrat with a grade that is too high or too low.
There are accessors for the attributes.
There are functions to increment / decrement the grade,
They throw exceptions when it's appropriate. Remember that incrementing a grade 3 will give you a grade 2 (1 is the highest).
The exceptions that are used inherit from std::exception, or from something derived from std::exception (i.e. they are catchable as std::exception & e).
There is a << operator to ostream overload that outputs the info of the Bureaucrat.

☑ Yes ☒ No

Ex01: Form up, maggots!

As usual, there has to be the main function that contains 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.

Form

There is a Makefile that compiles using the appropriate flags.
There is a Form class.
It has a name, a bool that indicates whether is it signed (at the beginning it's not), a grade required to sign it, and a grade required to execute it.

The names and grades are constant.
All these attributes are private and not protected.
The grades of the forms follow the same constraints as the Bureaucrat's (exceptions, 1 = highest 150 = lowest, and so forth).
There are accessors for the attributes and a << operator to ostream overload that displays the complete state of the Form.
There is a Form::beSigned() member function that works as described by the subject.
There is a Bureaucrat::signForm() function that works as described by the subject.

☑ Yes ☒ No

Ex02: No, you need form 28B, not 28C...

As usual, there has to be the main function that contains 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.

Forms that actually do something

There is a Makefile that compiles using the appropriate flags.
There are concrete forms that are conform to the specifications of the subject (required grades, names and actions).
They take only one parameter in their constructor, which is their target.
There is a Form::execute(Bureaucrat const & executor) method that works as specified by the subject.
Either this method is pure and the grade checks are implemented in each subclass, or this method performs the checks, then calls another method in derived class that only executes the action.
Both of these techniques are valid.
There is a Bureaucrat::executeForm(Form const & form) that works as specified by the subject.

☑ Yes ☒ No

Ex03: At least this beats coffee-making

As usual, there has to be the main function that contains 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.

Intern

There is a Makefile that compiles using the appropriate flags.
There is an Intern class.
It has a makeForm() function that works as specified by the subject.

☑ Yes ☒ No

Good dispatching

The makeForm() function should use some kind of array of pointers to member functions to handle the creation of Forms.
If it's using an unclean method, like if/elseif/elseif/else branchings, or some other ugly stuff like this, please count this as wrong.

☑ Yes ☒ No