42 Evals

Back to all evaluation sheets

Points earned
0

CPP Module 05

You should evaluate 1 student in this team

Introduction

Please follow the rules below:

✓ Remain polite, courteous, respectful, and constructive throughout the evaluation process. The community's well-being depends on it.

✓ Work with the student or group being evaluated to identify potential issues in their project. Take time to discuss and debate the problems identified.

✓ Understand that there may be differences in how peers interpret the project instructions and scope. Always keep an open mind and grade as honestly as possible. Pedagogy is effective only when peer evaluations are taken seriously.

Guidelines

Please follow the guidelines below:

✓ Only grade the work submitted to the Git repository of the evaluated student or group.

✓ Double-check that the Git repository belongs to the student(s) and that the project is the one expected. Ensure that git clone is used in an empty folder.

✓ Carefully verify that no malicious aliases are used to d grading non-official content.

Points earned
0

✓ If applicable, review any scripts used for testing or aut student.

✓ If you haven't completed the assignment you're evaluating, read the entire subject before starting the evaluation.

✓ Use the available flags to report an empty repository, a non-functioning program, a Norm error, or cheating. The evaluation process ends with a final grade of 0 (or -42 for cheating). However, except in cases of cheating, students are encouraged to review the work together to identify mistakes to avoid in the future.

✓ Remember that no segfaults or other unexpected program terminations will be tolerated during the evaluation. If this occurs, the final grade is 0. Use the appropriate flag.

✓ You should not need to edit any files except the configuration file, if it exists. If editing a file is necessary, explain the reasons to the evaluated student and ensure mutual agreement.

✓ Verify the absence of memory leaks. All memory allocated on the heap must be properly freed before the program ends.

✓ You may use tools like leaks, valgrind, or e_fence to check for memory leaks. If memory leaks are found, tick the appropriate flag.

Attachments

Please download the attachments below:

subject.pdf

Mandatory Part

Prerequisites

Points earned

0

If cheating is suspected, the evaluation stops here. Use
it. Take this decision calmly, wisely, and please, use this

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.

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

Points earned
0

There is a Makefile that compiles using the appropriate

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
grades of the forms follow the same constraints as the I
= highest 150 = lowest, and so forth). There are access
<< operator to ostream overload that displays the comp
There is a Form::beSigned() member function that work
subject. There is a Bureaucrat::signForm() function that
subject.

Points earned
0

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

Points earned
0

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

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

Points earned

0

Bonus Part

no bonus

no bonus

no bonus

Yes No

Ratings

OK Outstanding Empty Work

Incomplete Work Invalid Compilation

Cheat Crash Concerning Situations

Leaks Forbidden Functions

Points earned

0