encouraged to continue to discuss your work (even if you have not finished it) to identify any issues that may have caused this failure and avoid repeating the same mistake in the future.

- Remember that for the duration of the defense, no segfault, no other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag.
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.

Disclaimer

Please respect 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 person (or the group) evaluated the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.

- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer evaluation is conducted seriously.

Guidelines

You must compile with clang++, with -Wall -Wextra -Werror
As a reminder, this project is in C++98 and C++20 members 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 (except in a template)
- A Makefile compiles without flags and/or with something other than clang++

Any of these means that you must flag the project as Forbidden Function:
- Use of a "C" function (*alloc, *printf, free)
- Use of a function not allowed in the subject
- Use of "using namespace" or "friend"

- Use of an external library, or C++20 features

Attachments

 subject.pdf (https://cdn.intra.42.fr/pdf/pdf/26541/en.subject.pdf)

ex00

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox's form, do not grade this exercise.

ex00

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 too high/low.
There are getters for the attributes.
There are functions to increment / decrement the grade,
they throw exceptions when appropriate. Remember that incrementing a
grade to 3 gives you a grade 2 since 1 is the highest... The exceptions 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

As usual, there has to be a main function that contains enough test to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical form, do not grade this exercise.

ex01

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 name and grades are constant.
All these attributes are private and not protected. The
grades have the same constraints as in the Bureaucrat
(Exceptions, 1 = highest 150 = lowest, etc...).

There are getters 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

As usual, there has to be a main function that contains enough test to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical form, do not grade this exercise.

---

ex02

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 the 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 does the checks then calls another method that only runs the action and is pure in the base class, both of these techniques are valid.
There is a Bureaucrat::executeForm(Form const & form) that works as specified by the subject.

 Yes No

---

ex03

As usual, there has to be a main function that contains enough test to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical form, do not grade this exercise.

---

ex03

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 a worse
method, like if/elseif/elseif/else branchings, or some other ugly stuff like
this, please count this as wrong.

Yes No

Ratings

Don’t forget to check the flag corresponding to the defense

Ok Outstanding project

Empty work No author file Invalid compilation Norme Cheat Crash

Leaks Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation