



## Menu

- My projects
- Holy Graph
- List projects
- Available Cursus

Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

## SCALE FOR PROJECT CPP MODULE 05

You should evaluate 1 student in this team



Git repository

git@vogsphere-v2-bg.1:

### Introduction

- Only grade the work that is in the student or group's GiT repository.

- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases were used to fool you and make you evaluate something other than the content of the official repository.

- To avoid any surprises, carefully check that both the evaluating and the evaluated students have reviewed the possible scripts used to facilitate the grading.

- If the evaluating student has not completed that particular project yet, it is mandatory for this student to read the entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository, non-functioning program, a norm error, cheating etc. In these cases, the grading is over and the final grade is 0 (or -42 in case of cheating). However, with the exception of cheating, you are encouraged to continue to discuss your work (even if you have not finished it) in order 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 defence, 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](#)

### 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 Coplien's form, do not grade this exercise.*

#### ex00

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 Coplien's form, do not grade this exercise.*

#### ex01

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 Coplien's form, do not grade this exercise.*

#### ex02

There are concrete forms that 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 Coplien's form, do not grade this exercise.

### Good dispatching

The makeForm function should really 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

## ex03

There is an Intern class.  
It has a makeForm function that works as specified by the subject.

✓ Yes

✗ No

## ex04

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 Coplien's form, do not grade this exercise.

## ex04

There is an OfficeBlock class.  
It has pointers to one Intern,  
one Bureaucrat that's the "signing" one, and one Bureaucrat that's the "executing" one.  
It can be constructed either with all three or with nothing.  
It has functions to set a new intern or new bureaucrats.  
It has a doBureaucracy function that works as specified by the subject.  
If the three members are not all set, the doBureaucracy function can not work.

✓ Yes

✗ No

### Good exceptions

Rate the specificity of the exceptions thrown when using doBureaucracy here.  
0 when there are no exceptions at all, 5 when there is one exception class by error type.

Rate it from 0 (failed) through 5 (excellent)



## ex05

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 Coplien's form, do not grade this exercise.

## ex05

There is a CentralBureaucracy class.  
It has 20 office blocks.

It can be created without parameters.  
You can "feed" Bureaucrats to it, and they are used to fill the office blocks. If all the blocks are filled, new Bureaucrats are either rejected or stored in a waiting list of some sort.  
Interns required to fill the blocks are generated automatically.  
It is possible to queue target names in the object.  
There is a doBureaucracy function that does some random bureaucracy to each target that was queued up, using the officeblocks it has created.

✓ 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