Menu
My projects
Holy Graph
List projects
Available Cursus
Your projects
Born2beroot
push_swap

Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defenses 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 04

You should evaluate 1 student in this team

Git repository

`git@vogsphere-v2.42ly<`

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 before starting the defense.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, norm error, cheating etc. In these cases,
the grading is over and the final grade is 0 (or -42 in case of
cheating). However, except for cheating, you are
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

ex00

As usual, there has to be the 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 canonical class form, do not grade this exercise.

Thorough testing

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

Yes / No

inheritants

They are at least two classes that inherit from animal.
Cat and Dog.
The constructor and destructor outputs must have clear outputs.
Ask the student about constructor and destructor orders.

Yes / No

Easy subclass

The attribute type is set to the good value at init for every animal.
Cat must use "Cat" and Dog must use "Dog".

Yes / No

Animal

Using makeSound() function always called the appropriate makeSound() function
makeSound() should be virtual ! Look the code.
virtual void makeSound() const
return value is not important but virtual is mandatory.

there should be an example with a WrongAnimal and WrongCat that doesn't use virtual.
The WrongCat must output the WrongCat makeSound() only when used as a wrongCat.

Yes / No

ex01

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 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 in Animal class.
The brain class has specific output upon creation and deletion.

Yes / No

Concrete Brain

The copy operation of Cat and Dog should create a deep copy.
test something like:
Dog basic;
{
Dog tmp = basic
}
if the copy is not deep tmp and basic will use the same Brain.
And tmp will delete the Brain at the end of the scope.
the copy constructor should do a deep copy too.
That's why a clean implementation of orthodox canonical form will save you hours of pain.

Yes / No

Destructor chaining

The destructors in Animal and its derived classes are virtual.
Ask an explanation of what will happens without virtual.
Test it.

Yes / No

ex02

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 canonical class form, do not grade this exercise.

Abstract

The Animal class is present and is exactly like
the one in the subject.
The Animal::makeSound function is pure virtual.
something 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.

Yes / No

Concrete Animal

Class Cat and Dog are still present and work exactly like in ex02.

Yes / No

Assignment and copy

The copy and assignation behaviors of the Cat and Dog are like the subject required.
That means deep copy, you need to create a new Brain for the Dog or cat.
Check that te canonical form is really implemented (if, no empty copy operators...)
nothing should be public without reason.
Overall this code is very simple so it needs to be clean!

Yes / No

ex03

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 canonical form, do not grade this exercise.

Interfaces

The ICharacter and IMateriaSource interfaces are present and are
exactly like in the subject.

Yes / No

Source

The MateriaSource class is present and implements IMateriaSource. The
member functions work as intended.

Yes / No

Concrete materia

There are concrete Ice and Cure classes that inherit from AMateria Their
clone() method is correctly implemented. Their outputs are correct.
AMateria class is still abstract (clone is pure).
virtual ~AMateria() is present.
AMateria contains a protected string attribute to store the type.

Yes / No

Character

The Character class is present and implements ICharacter. It has
an inventory of 4 materias.
The member functions are implemented as the subject requires.
The copy and assignation of a Character are implemented as required => deep copy

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