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 04 (HTTPS://PROJECTS.INTRA.42.FR/PROJECTS/CPP-MODULE-04)

You should evaluate 1 student in this team

Git repository

```
git@vogsphere.msk.21-school.ru:vogsphere/intra-uuid-3a2fdec7-0117-4fa
```

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 (https://cdn.intra.42.fr/pdf/pdf/25937/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 Coplien's form, do not grade this exercise.

Thorough testing

There are tests in the main with derived classes other than Peon, and
everything works well with them.

 Yes No

I want sheeps !

The Victim can getPolymorphed() const, with the correct output. The
Sorcerer can polymorph(Victim const &) const.

 Yes No

Destructor chaining

The destructors in Victim and derived are virtual.

 Yes No

Easy subclass

There is a Peon class that inherits publicly from Victim. It has
the correct outputs.

 Yes No

Victim

There is a Victim class. It has a name. The required outputs
on construction and destruction are present.
The required overload of operator << to ostream is present and works
correctly

□ Yes □ No

Sorcerer

There is a Sorcerer class. It has a name and a title. It has
a constructor with name and title.
It cannot be instanciated without parameters.
That means either the default constructor must be private, or it must be
declared but non-implemented, to comply with Coplien's form.
The required outputs on construction and destruction are present.
The required overload of operator << to ostream is present and works correctly.

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

Concrete enemies

There are concrete SuperMutant and RadScorpion enemies (That inherit
from Enemy, obviously)
They have the required attributes.
The SuperMutant
has the required overload of takeDamage() and it works as required.

□ Yes □ No

Character

There is a Character class. It has the attributes required by
the subject: name, AP, pointer to AWeapon.
It has the required AP behavior: 40 on start, it looses X AP on attack
depending on the weapon, and recovers 10 AP
with recoverAP up to a maximum of
40. attack(...) fails if there aren't enough
AP.

□ Yes □ No

Concrete weapons

There are concrete PlasmaRifle and PowerFirst weapons. (So, they
inherit from AWeapon)
They have the attributes and attack() outputs specified
by the subject.

□ Yes □ No

Utility and output

The equip() and attack() functions work as required. The << overload
works as required.

□ Yes □ No

Destructor chaining 2

The destructors in AWeapon and its derived classes are virtual.

☐ Yes ☐ No

Thorough testing

There are tests in the main with more derived weapons and more derived enemies.

☐ Yes ☐ No

Destructor chaining AGAIN

The destructors in Enemy and its derived classes are virtual.

☐ Yes ☐ No

Enemy

There is an Enemy class. It has the attributes required by the subject: type, number of HP
Its member functions are implemented coherently.
It has the required check in takeDamage to prevent going under 0 HP.

☐ Yes ☐ No

Weapon

There is an AWeapon class. It is abstract (attack() must be a pure virtual function).
It has the attributes required by the subject : name, damage, AP cost.
Its member functions are implemented coherently

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

Interfaces

The ISquad and ISpaceMarine interfaces are present and are exactly like the ones in the subject.

☐ Yes ☐ No

Concrete squad

The Squad class is present and inherits from ISquad Its member functions work as required.
Its destructor destroys the contained units.

☐ Yes ☐ No

Concrete units

The TacticalMarine and AssaultTerminator classes are present and inherit from ISpaceMarine.
Their member functions work as required.

☐ Yes ☐ No

Assignment and copy

The copy and assignation behaviours of the Squad are as the subject required.
That means deep copy, and upon assignation, exiting units must be
destroyed before they are replaced.

□ 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 Coplien's 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.

□ 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.

□ Yes □ No

Materia base

There is an AMateria class. It has a type. It's abstract (clone is pure).
The XP system is implemented as the subject requires.

□ Yes □ No

Assignation and copy

The copy and assignation of a Character are implemented as required (=
deep copy, very much like the previous exercise).

□ 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.

Basics

The IAsteroid and IMiningLaser interfaces are present. Concrete
Asteroids and MiningLasers are implemented.

□ Yes □ No

DD's patcher !

The mine/beMined dispatch mechanism works as required. In theory,
there should be a beMined(StripMiner *) and a beMined(DeepCoreMiner*),
and the mine() method should call beMined passing "this" as parameter,
which would dispatch the call to a method that depends on the type of the
asteroid (subtype polymorphism) and the type of the laser (adhoc polymorphism).
Basically the double-dispatcher design pattern, just a bit dumber.
Now the clever bit: if the student tries to pass off a technique that
uses typeid, dynamic_cast, the names of the lasers/asteroids, etc. to
select the output, MARK THE WHOLE PROJECT AS CHEAT and leave it at that,
because it is EXPLICTLY forbidden by the subject.

□ Yes □ No

Ratings

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

□ Ok □ Outstanding project

□ Empty work □ No author file W Invalid compilation □ Norme □ Cheat d Crash □ Leaks

l Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation