42 Evals

Back to all evaluation sheets

CPP Module 03

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 well-being of the community depends on it.

✓ - Identify with the student or group whose work is being evaluated the possible dysfunctions in their project. Take the time to discuss and debate the problems that may have been identified.

✓ - You must consider that there might be some differences in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade them as honestly as possible. The pedagogy is useful only if the peer-evaluation is done seriously.

Guidelines

Please follow the guidelines below:

✓ - Only grade the work that was turned in to the Git repository of the evaluated student or group.

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

✓ - Check carefully that no malicious aliases were used to fool you into evaluating something that is not the content of the official repository.

✓ - To avoid any surprises and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).

✓ - If you have not completed the assignment you are going to evaluate, you must read the entire subject prior to starting the evaluation process.

✓ - Use the available flags to report an empty repository, a non-functioning program, a Norm error, cheating, and so forth. In these cases, the evaluation process ends and the final grade is 0, or -42 in the case of cheating. However, except for cheating, students are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated in the future.

✓ - Remember that for the duration of the defense, no segfaults or other unexpected, premature, or uncontrolled terminations of the program will be tolerated, 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 explain 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.

Attachments

Please download the attachments below:

subject.pdf

Mandatory Part

Prerequisites

If cheating is suspected, the evaluation stops here. Use the "Cheat" flag to report it. Take this decision calmly, wisely, and please, use this button with caution.

- ✓ 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

Exercise 00: Annnnnd... ACTION!

As usual, there should be enough tests to prove that the program works as requested. If there are none, do not grade this exercise.

- ✓ There is a ClapTrap class. It has all the following private attributes:
- ✓ - name
- ✓ - hit points
- ✓ - energy points
- ✓ - attack damage
- ✓ These attributes are initialized to the requested values.

Yes No

Member Functions

The following member functions are present and function as specified:

- ✓ - attack()
- ✓ - takeDamage()
- ✓ - beRepaired()

Yes No

Exercise 01: Serena, My Love!

As usual, there should be enough tests to prove that the program works as requested. If there are none, do not grade this exercise.

- ✓ Class and Attributes
- ✓ There is a ScavTrap class. ScavTrap publicly inherits from the Claptrap class. It does not redeclare the attributes. The attributes of the ClapTrap class are

now protected instead of private. The attributes are initialized to the requested values.

Yes No

Member Functions

The following member functions are present and functional:

- ✓ - attack()
- ✓ - takeDamage() (inherited)
- ✓ - beRepaired() (inherited)
- ✓ The messages from the constructor, destructor, and attack() function must be different from those of the ClapTrap.

Yes No

Construction and Destruction

Construction and Destruction

- ✓ ScavTrap must have a constructor and destructor with specific messages. Their proper implementation must be demonstrated by a sequence of calls in the expected order: if you create a ScavTrap, the message from the ClapTrap constructor should be displayed first, followed by that of the ScavTrap. Conversely, if you delete a ScavTrap, the message from the ScavTrap destructor should be displayed first, followed by that of the ClapTrap.

Yes No

Special Feature

Special Feature

✓ ScavTrap has a guardGate() function that displays a message on the standard output. ScavTrap also has an attack() function that displays a message different from that of the ClapTrap on the standard output.

Yes No

Exercise 02: Assembly Line Work

As usual, there should be enough tests to prove that the program works as requested. If there are none, do not grade this exercise.

✓ Class and Attributes

✓ There is a FragTrap class that publicly inherits from ClapTrap. Attributes should not be redeclared without reason

Yes No

Construction and Destruction

Construction and Destruction

✓ FragTrap must have a constructor and destructor with specific messages. Their proper implementation must be demonstrated by a sequence of calls in the expected order: if you create a FragTrap, the message from the ClapTrap constructor should be displayed first, followed by that of the FragTrap. Conversely, if you delete a FragTrap, the message from the FragTrap destructor should be displayed first, followed by that of the ClapTrap.

Yes No

Special Feature

Special Feature

✓ FragTrap has a highFivesGuys() function that displays a message on the standard output.

Yes No

Exercise 03: Ok, This Is Getting Weird

As usual, there should be enough tests to prove that the program works as requested. If there are none, do not grade this exercise.

✓ The Ultimate Weirdness of C++

✓ There is a DiamondTrap class. It inherits from both FragTrap and ScavTrap. It defines attributes with the requested values. It uses virtual inheritance to avoid the pitfalls of diamond inheritance.

Yes No

Choose Wisely...

Choose Wisely...

✓ The DiamondTrap class uses the attack() function of Scavtrap. It has the special functions of both its parents. DiamondTrap has a private member std::string name. The whoAmI() function has access to both name and ClapTrap::name.

Yes No

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

Cannot Support/Explain code