SCALE FOR PROJECT CPP MODULE 03

Introduction

Please comply with 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 student or group whose work is 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 and only if the peer-evaluation is done seriously.

Guidelines

- Only grade the work that was turned in 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 was used to fool you and make you evaluate 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 have to 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 case of cheating. However, except for cheating, student 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.

- 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.

Preliminary tests

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.

Prerequisites

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 " or the "friend" keyword.
- Use of an external library, or features from versions other than C++98.

☑ Yes ☒ No

Ex00: Aaaaand... OPEN!

As usual, there has to be enough tests to prove the program works as required. If there isn't, do not grade this exercise.

Class and attributes

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

☑ Yes ☒ No

Member functions

The class has following member functions and they work as specified:
- attack
- takeDamage
- beRepaired

☑ Yes ☒ No

Exercise 01: Serena, my love!

As usual, there has to be enough tests to prove the program works as required. If there isn't, do not grade this exercise.

Class and attributes

There is a ScavTrap class.
The ScavTrap publicly inherits from the ClapTrap class.
It does not re-declare any attributes.
The ClapTrap attributes are now protected instead of private.
The attributes are initialized to the required values.

☑ Yes ☒ No

Member functions

The class has following member functions and they work as specified:
- attack
- takeDamage (inherited)
- beRepaired (inherited)
The outputs of the constructor, destructor, and the attack() function must be different from the ClapTrap's.

☑ Yes ☒ No

Construction and destruction

There must be a constructor and a destructor for the ScavTrap displaying specific messages. They must be implemented so that they are called in the correct order when used. This means if you create a ScavTrap, it must first display the ClapTrap's message then the ScavTrap's. If you delete a ScavTrap, it must display the ScavTrap's message first, then the ClapTrap's.

☑ Yes ☒ No

Special feature

ScavTrap has a guardGate() function that displays a message on the standard output. It has also an attack() function that displays a short message on the standard output, which must be different from the original ClapTrap message.

☑ Yes ☒ No

Ex02: Repetitive work

As usual, there has to be enough tests to prove the program works as required. If there isn't, do not grade this exercise.

Class and attributes

There is a FragTrap class that publicly inherits from ClapTrap.
Attributes must not be re-declared without reasons.

☑ Yes ☒ No

Construction and destruction

There must be a constructor and a destructor for the FragTrap displaying specific messages. They must be implemented so that they are called in the correct order when used. This means if you create a FragTrap, it must first display the ClapTrap's message then the FragTrap's. If you delete a FragTrap, it must display the FragTrap's message first, then the ClapTrap's.

☑ Yes ☒ No

Special feature

There is a highFivesGuys() function that displays a message on the standard output.

☑ Yes ☒ No

Ex03: Now it is weird!

As usual, there has to be enough tests to prove the program works as required. If there isn't, do not grade this exercise.

Ultimate C++ weird feature

There is a DiamondTrap class.
It inherits from both the FragTrap and the ScavTrap.
The attributes are set to the appropriate values.
It uses virtual inheritance to avoid the pitfalls of diamond inheritance.

☑ Yes ☒ No

Choose wisely...

The DiamondTrap uses the attack() method of the Scavtrap.
It has the special functions of both its parents.
The DiamondTrap has a private std::string name member.
The function whoAmI() can display both name and clapTrap::name.

☑ Yes ☒ No