SCALE FOR PROJECT CPP MODULE 01

You should evaluate 1 student in this team

Git repository

```
git@vogsphere.42istanbul.com.trivogsphere/intra-uuid-3a26f499-1297-4
```

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.

Attachments

subject.pdf

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

Yes | No

Exercise 00: Braiiiiiiinnnzzzz

The goal of this exercise is to understand how to allocate memory in C++.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

Zombie Class

There is a Zombie Class.
It has a private name attribute.
It has at least a constructor.
It has a member function announce(void) that prints: "<name>: Braiiiiiiinnnzzzz..".
The destructor prints a debug message that includes the name of the zombie.

Yes | No

newZombie

There is a newZombie() function prototyped as: [Zombie* newZombie(std::string name);]
It should allocate a Zombie on the heap and return it.
Ideally, it should call the constructor that takes a string and initializes
the name.
The exercise should be marked as correct if the Zombie can announce itself
with the name passed to the function.
There are tests to prove everything works.
The zombie is deleted correctly before the end of the program.

Yes | No

randomChump

There is a randomChump() function prototyped as: [void randomChump(std::string name);]
It should create a Zombie on the stack, and make it announce itself.
Ideally the zombie should be allocated on the stack (so implicitly deleted
at the end of the function). It can also be allocated on the heap and then
explicitly deleted.
The student must justify their choices.
There are tests to prove everything works.

Yes | No

Exercise 01: Moar brainz!

The goal of this exercise is to allocate a number of objects at the same time using new[], initialize them, and to properly delete them.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

zombieHorde

The Zombie Class has a default constructor.
There is a zombieHorde() function prototyped as: [Zombie* zombieHorde(int N, std::string name);]
It allocates N zombies on the heap explicitly using new[].
After the allocation, there is an initialization of the objects to set their name.
It returns a pointer to the first zombie.
There are enough tests in the main to prove the previous points.
Like: calling announce() on all the zombies.
Last, all the zombies should be deleted at the same time in the main.

Yes | No

Exercise 02: HI THIS IS BRAIN

Demystify references! Demystify references! Demystify references! Demystify references! Demystify references! Demystify references! Demystify references! Demystify references! Demystify references! Demystify references!

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

HI THIS IS BRAIN

There is a string containing "HI THIS IS BRAIN".
stringPTR is a pointer to the string.
stringREF is a reference to the string.
The address of the string is displayed using the string variable, the
stringPTR and the stringREF.
The variable content is displayed using the stringPTR and the stringREF.

Yes | No

Exercise 03: Unnecessary violence

The objective of this exercise is to understand that pointers and references present some small differences that make them less or more appropriate depending on the use and the lifecycle of the object used.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

Weapon

There is a Weapon class that has a type string, a getType() and a setType().
The getType() function returns a const reference to the type string.

Yes | No

HumanA and HumanB

HumanA can have a reference or a pointer to the Weapon.
Ideally, it should be implemented as a reference, since the Weapon exists
from creation until destruction, and never changes.
HumanB must have a pointer to a Weapon since the field is not set at
creation time, and the weapon can be NULL.

Yes | No

Exercise 04: Sed is for losers

Thanks to this exercise, the student should have gotten familiar with fstream and ostream.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

Exercise 04

There is a function replace (or other name) that works as specified in the
subject.
The error management is efficient: try to pass a file that does not exist,
change the permissions, pass it empty, etc.
If you can find an error that isn't handled, and isn't completely esoteric,
no points for this exercise.
The program must read from the file using an fstream or equivalent,
and write using an ostream or equivalent.
The implementation of the function should be done using functions from
std::string, no by reading the string character by character.
This is not C anymore!

Yes | No

Exercise 05: Harl 2.0

The goal of this exercise is to use pointers to class member functions. Also, this is the opportunity to discover the different log levels.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

Our beloved Harl

There is a class Harl with at least the 5 functions required in the subject.
The function complain() executes the other functions using a pointer to them.
Ideally, the student should have implemented a way of matching the different
strings corresponding to the log level to the pointers of the corresponding
member function.
If the implementation is different but the exercise works, you should mark
it as valid. The only thing that is not allowed is using a ugly if/elseif/else.
The student could have chosen to change the message Harl displays or to
display the examples given in the subject, both are valid.

Yes | No

Exercise 06: Harl filter

Now that you are experienced coders, you should use new instruction types, statements, loops, etc. The goal of this last exercise is to make you discover the switch statement.

Makefile and tests

There is a Makefile that compiles using the appropriate flags.
There is at least a main to test the exercise.

Yes | No

Switching Harl Off

The program harlFilter takes as argument any of the log levels ("DEBUG",
"INFO", "WARNING" or "ERROR"). It should then display just the messages
that are at the same level or above (DEBUG < INFO < WARNING < ERROR). This
must be implemented using a switch statement with a default case.
Once again, no if/elseif/else anymore please.

Yes | No

Ratings

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

Ok | Outstanding project

Empty work | Incomplete work | Invalid compilation | Cheat | Crash

Concerning situation | Leaks | Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation