Menu

My projects

Hide Graph

List projects

Available Cursus

Your projects

cub3d

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 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 01

You should evaluate 1 student in this team

Git repository

`git@vogsphere.42lisboa.com:vogsphere/intra-uuid-b86edca9-d79`

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

ex00

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

Makefile and main

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

Yes / No

Zombie Class

There is a Zombie Class.
It has a name attribute.
It has at least a default constructor.
It has a member function announce(void) that prints: " BraiiiiiinnnzzZ... "
The destructor prints a 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.
The main contains tests to prove so.
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 his choice.
The main contains tests to prove so.

Yes / No

ex01

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

Makefile and main

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

Yes / No

zombieHorde

The 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.
Ex: calling announce() on all the zombies.
Last, all the zombies should be deleted at the same time in the main.

Yes / No

ex02

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

Makefile and main

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

Yes / No

ex04

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

Yes / No

ex03

The objective of this exercise it to understand that pointers and references present some small differences that make them be more appropriated depending on the use and the lifecycle of the object we are going to use.

Makefile and main

There is a Makefile that compiles using the appropriate flags.
There is 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 because the field is not
set at creation time, and the weapon can be NULL.

Yes / No

ex04

Doing this exercise you should have gotten familiar with ifstream and ofstream.

Makefile and main

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

Yes / No

ex04

There is a function replace that works as specify in the subject.
The error management is good: 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 ifstream or equivalent,
and write using an ofstream 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

ex05

The goal of this exercise is to make you use pointers to class member functions. Also, we consider it a great moment to introduce you to the different log levels.

Makefile and main

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

Yes / No

Our beloved Karen

There is a class Karen 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 to have a if/elseif/else.
The student could have chosen to change the message Karen displays or to display
the examples given in the subject, both are valid.

Yes / No

ex06

Now that you are grown up coders, you should start to use new instruction types, statements, loops, etc. The goal of this last exercise is to make you discover the SWITCH.

Makefile and main

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

Yes / No

Switching Karen Off

The program karenFilter takes as argument any of the log levels: "DEBUG", "INFO",
"WARNING" and "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. Once again, no if/elseif/else anymore please.
The switch should have a default case.

Yes / No

Ratings

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

Ok

Empty work / Incomplete work / Invalid compilation / Cheat

Crash / Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation