SCALE FOR PROJECT CPP MODULE 08

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) are NOT expected.
The purpose of this module is to use the STL. Then, using the containers and the algorithms is authorized.

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: Easy find

As usual, there has to be the main function that contains enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

Template function

There is a templated function easyFind(T, int) that does what the subject requires.
It HAS to use STL algorithms.
If it does not (like manual search using iterators for example), count it as wrong.

☑ Yes ☒ No

Ex01: Span

As usual, there has to be the main function that contains enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

Class and member functions

There is a class that complies with the constraints of the subject.
Its member functions use STL algorithms to find their result as much as possible. Finding the shortest span can't be done only by subtracting the two lowest numbers (take a closer look at the subject example).

☑ Yes ☒ No

Improved addNumber function

There's a way to add numbers that's more practical than calling the addNumber() function repeatedly.

☑ Yes ☒ No

Ex02: Mutated abomination

As usual, there has to be the main function that contains enough tests to prove the program works as expected. If there isn't, do not grade this exercise. If any non-interface class is not in orthodox canonical class form, do not grade this exercise.

MutantStack class

There is a MutantStack class that inherits from std::stack and offers all of its member functions.
It has an iterator. Also, it is possible to do at least the operations shown in the subject's examples using iterators.

☑ Yes ☒ No

Better tests

There is a at least a main() function that has more tests than the ones from the subject.

☑ Yes ☒ No