# SCALE FOR PROJECT PISCINE CPP (/PROJECTS/PISCINE-CPP) / DAY 08 (/PROJECTS/42-PISCINE-C-FORMATION-PISCINE-CPP-DAY-08)

## Introduction

The subject of this project is rather vague and leaves a lot to the user's choice. This is INTENDED. The questions in this grading scale, however, are very focused and concentrate on what we think is the core of each exercise, what we want you to grasp. So we would like you to do the same : You can and should tolerate moderate deviations in filenames, function names, etc ... as long as the exercise basically works as intended. Of course, in case the student you are grading really strayed too far, you should not grade the exercise in question at all. We leave it to your good judgement to determine what constitutes "straying too far".

The usual obvious rules apply : Only grade what's on the git repository of the student, don't be a dick, and basically be the grader you would like to have grading you.

Do NOT stop grading when an exercise is wrong.

## Guidelines

You must compile with clang++, with -Wall -Wextra -Werror

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++
- A non-interface class is not in Coplien's form

Any of these means that you must flag the project as Cheat:
- Use of a "C" function (*alloc, *printf, free)
- Use of a function not allowed in the subject
- Use of "using namespace" or "friend" (Unless explictly allowed in the subject)
- Use of an external library, or C++11 features (Unless explictly allowed in the subject)
- Use of a C-style cast

## Attachments

Subject (https://cdn.intra.42.fr/pdf/pdf/2065/d08.en.pdf)

## ex00

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

**ex00**

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

Yes No

## ex01

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

**Better addNumber**

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

Yes No

**ex01**

There is a class that respects the constraints of the subject.
Its member functions use STL algorithms to find their result, as much as possible.

Yes No

## ex02

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

**ex02**

There is a MutantStack class that inherits from std::stack (Or is composed of one, dealer's choice), and offers all of its member functions.
It has an iterator, and it is possible to do at least the operations in the subject's example with it.

✓ Yes ✗ No

**Better tests**

There is a test main() function that has more tests than the one in the subject.

✓ Yes ✗ No

## ex03

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

**ex03**

There is a program that can interpret a Brainfuck-like language.
It functions correctly (The student has to provide test files to prove it)
It works in the way specified by the subject, that is, it has a set of Instruction classes that inherit from a common interface or class, it reads from the file, creating one such Instruction object and storing it in an appropriate container, then once the file is fully read, it executes the Instructions.
It has to use STL containers and algorithms.
If the way it works deviates too much from what the subject requires, count as wrong.

✓ Yes ✗ No

## ex04

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

**ex04**

The program works as the subject requires:
- First, converts the expression to a set of Token-derived objects
- Converts the expression to postfix (aka Reverse Polish) notation
- Evaluates the expression while outputting every single step, as in the subject's example.
Errors are handled appropriately.
STL algorithms are used in a reasonable enough amount.

✓ Yes ✗ No

## Ratings

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

✔ Ok

Empty work Incomplete work No author file Invalid compilation

Norme Cheat Crash Forbidden function

## Conclusion

**Leave a comment on this evaluation**

Preview!!!