Introduction

In this assignment you will overload `operator new()` and `operator delete()` in such a way that the number of dynamic allocations are counted, and at the end of the program a small report is printed to the terminal. This report shows how many allocations where done in total and how many allocations where active at the same time. If there were allocations left at the end, then the report will also show how many such allocations remain (not having been deallocated). This report should *only* be provided if allocations happened.

To make this work across several different translation units you are going to need to have shared counters stored in *static memory*. You are also going to need some way to produce the report *after* `main()` is finnished.

There are given files `other.h`, `other.cc` and `main.cc`, these are just some simple tests that perform various allocations and deallocations within two *different* translation units. Therefore it is important that the includes of the files are not modified. Make sure to compile *all* implementation files (`.cc`).

All code you are going to write should go in `tracker.h` and `tracker.cc`. The remainder of this document consists of hints on how to implement this, however note that you should try to do this on your own before reading these hints.

Potential implementation details

To implement this, make three global counters: `allocations`, `total_allocations`, and `max_allocations`, note that these must be accessible from all translation units (might have to use one of `inline`, `extern` or `static`).

In `operator new()`, call `std::malloc()` then increase the `allocations` counter and the `total_allocations` counter. If `allocations` exceeded `max_allocations`, then update `max_allocations` to `total_allocations`. These modifications should only happen if the allocation happened (i.e. if `std::malloc()` returned a non-`nullptr`).

In `operator delete()`, frees the memory using `std::free()` and then decreases `allocations`, but only if the deleted memory is non-`nullptr`.

Lastly we must make the report appear at the end of program execution. This means it has to be printed in a destructor for an object that gets destroyed during program termination. So implement a class who prints the report in its destructor and create an instance of it in static memory.