Potential Pitfalls

Corresponding explicit-instantiation declarations and definitions

To realize a reduction in object-code size for individual translation units and yet still be able to link all valid programs successfully into a well-formed program, four moving parts have to be brought together correctly.

1. Each general template, `C<T>`, whose object code bloat is to be ~~optimized~~ must be declared within some designated component's header file, `c.h`.
2. ~~The specific definition of each `C<T>` relevant to an explicit specialization being optimized — including general, partial-specialization, and full-specialization definitions — must appear in the header file prior to its corresponding explicit-instantiation declaration.~~
3. Each explicit-instantiation declaration for each specialization of each separate top-level — i.e., class, function, or variable — template must appear in the component's `.h` file ~~*after* the corresponding general template declaration and the relevant general, partial-specialization, or full-specialization definition, but, in practice, always after *all* such definitions, not just the relevant one~~.
4. Each template specialization having an explicit-instantiation declaration in the header file must have a corresponding explicit-instantiation definition in the component's implementation file, `c.cpp`.

Absent items (1) and (2), clients would have no way to safely separate out the usability and inlineability of the template definitions yet consolidate the otherwise redundantly generated object-level definitions within just a single translation unit. Moreover, failing to provide the relevant definition would mean that any clients using one of these specializations would either fail to compile or, arguably worse, pick up the general definitions when a more specialized definition was intended, likely resulting in an ill-formed program.

Failing item (3), the object code for that particular specialization of that template will be generated locally in the client's translation unit as usual, negating any benefits with respect to local object-code size, irrespective of what is specified in the `c.cpp` file.

Finally, unless we provide a matching explicit-instantiation definition in the `c.cpp` file for each and every corresponding explicit-instantiation declaration in the `c.h` file as in item (4), our optimization attempts might well result in a library component that compiles, links, and even passes some unit tests but, when released to our clients, fails to link. Additionally, any explicit-instantiation definition in the `c.cpp` file that is not accompanied by a corresponding