Accidentally making matters worse

When making the decision to explicitly instantiate common specializations of popular templates within some designated object file, it is important to consider that not all programs necessarily need every (or even any) such instantiation. Classes that have many member functions but typically use only a few require special attention.

For such classes, it might be beneficial to explicitly instantiate individual member functions instead of the entire class template. However, selecting *which* member functions to explicitly instantiate and with *which* template arguments they should be instantiated without carefully measuring the effect on the overall object size might result in not only overall pessimization, but also ~~to~~ an unnecessary maintenance burden. Finally, remember that one might need to explicitly tell the linker to strip unused sections resulting, for example, from forced instantiation of common template specializations, to avoid inadvertently bloating executables, which could adversely affect load times.

Annoyances

No good place to put definitions for unrelated classes

When we consider the implications of **physical dependency**,[5,6] determining in which component to deposit the specialized definitions can be problematic. For example, consider a codebase implementing a core library that provides both a nontemplated `String` class and a `Vector` container class template. These fundamentally unrelated entities would ideally live in separate physical components (i.e., `.h`/`.cpp` pairs), neither of which depends physically on the other. That is, an application using just one of these components could be compiled, linked, tested, and deployed entirely independently of the other. Now, consider a large codebase that makes heavy use of `Vector<String>`: In what component should the object-code-level definitions for the `Vector<String>` specialization reside?[7] There are two obvious alternatives.

1. `vector` — In this case, `vector.h` would hold **`extern template class`** `Vector<String>;` — the explicit-instantiation declaration. `vector.cpp` would hold **`template class`** `Vector<String>;` — the explicit-instantiation definition. With this approach, we would create a physical dependency of the `vector` component on `string`. Any client program wanting to use a `Vector` would also depend on `string` regardless of whether it was needed.

[5] See **lakos96**.

[6] See **lakos20**.

[7] Note that the problem of determining in which component to instantiate the object-level implementation of a template for a user-defined type is similar to that of specializing an arbitrary **user-defined** trait for a user-defined type.