

VISUAL C++ PROBLEM RE LINKER ERRORS WHEN USING TEMPLATED CLASSES AND/OR FUNCTIONS

M. L. Malone, Nov. 03

Based on information from T. Atanasio at Salisbury University

Thanks to my counterparts at Salisbury University, I now know the problem concerning linking modules that contain source code for template classes and/or template functions in Visual C++ 6.0. According to the folks at SU, the SGI compiler that uses an ATT cfront approach handles this gracefully. Visual C++, Borland C++, and GCC do not.

Good programming requires that the specification of a class be within a header file with its implementation in a .cpp file. This supports information hiding and lots of other good software engineering principles! This works in most cases—but it fails when templated classes and/or functions are used in Visual C++ (at least in version 6.0).

According to Atanasio, “The problem is that the compiler cannot determine template instantiations until it sees a request for one. By that time, it has ‘lost’ the definition of the template class (or function) because it has already “compiled” the implementation file separately. Somehow it needs to have the implementation available in every compilation unit that instantiates template objects.

He continues, “The textbook approach is a very standard one, namely pretend that you have separated interface and implementation by having .h and .cpp files, but #include the .cpp at the tail of the .h. In effect, we make one big file that contains both interface and implementation.”

Here are the rules that Atanasio teaches his students at SU:

1) Always guard a header file

```
#ifndef FOO_H
#define FOO_H
.... usual stuff ....
#endif
```

whether it contains template code or not. This helps to avoid "multiple definition" errors. (We always do this!)

2) Never mix template and non-template code in the same file. If a file has any template code, it should not also have non-template code and vice-versa. (Aha!)

3) For template code files, place the implementation code at the end of the interface (specification) code in the same file. (Sheesh!)

4) Do not separately compile any template code files. For the Visual-C++ compiler this means:

- a) *.cpp files that contain template code should not be added to the project. If they are added, then they will get separately compiled - disaster.
- b) *.cpp files that do not contain template code should be added to the project so they will be separately compiled.

Let's hope that future editions of C++ compilers will find a way to handle this! Until then...

>> mlm