Letter to the Editors

This note was written in June 1998 and sent to various people I knew to be active in the Fortran community. I also posted it to comp.lang.fortran where a good discussion took place regarding this issue. I would urge people to look at the archives from that newsgroup for late June and July. While I learned a great deal from that discussion, nothing I read changed the fundamental concerns I express below.

X3J3, the standards committee developing Fortran 2000, has been tasked by its parent organization with adding object-oriented capability to Fortran 2000. This is something that I doubt has received much press in the Fortran community and even less debate. Personally, after some intial enthusiasm for the OOPs paradigm (though not for some of the OOPs languages other there), I am finding myself increasingly questioning of the utility of object-oriented programming for scientific applications, largely based on my observations of the enormous difficulties people have encountered in doing scientific programming in C++ (in particular the very steep learning curve involved, the difficulty of maintaining the C++ codes, and the difficulties of reusing C++ code). The problems of using an OOP paradigm for scientific computing may yet be solved down the road, but I would prefer Fortran not add this paradigm to its already hefty bulk until this paradigm has had a chance to either mature in a truly useful one for scientific programming or to fade into the woodwork.

To back up my concerns with opinions other than my own, below is an excerpt from a recent article by William Press and Saul Teukolsky, two of the authors of the books *Numerical Recipes*. Some of what they discuss is specific to C++, but much of it is related more to object-oriented programming than any specific implementation of that paradigm.

I would be curious for other people's opinions, particularly those who have had experience with non-trivial numerical computer codes written in C++ or Java. I would also urge people to contact X3J3 and express an opinion.

John K. Prentice
Quetzal Computational Associates, Inc.
3455 Main Avenue, Suite 4
Durango, CO 81301
USA
Tel.: +1 970 382 8979
Fax: +1 970 382 8981
E-mail: john@quetzalcoatl.com

From "Numerical recipes: Does the paradigm have a future?" by William H. Press and Saul A. Teukolsky, *Computers in Physics* **11**(5) (1997), 416–424.

The following is excerpted from pages 422–423:

And what about C++?

Indeed, what about C++? This language would seem to meet all our requirements: It allows arbitrary high-level constructions through the mechanisms of a class library, yet its underlying C syntax is even more primitive, and closer to the machine, than old Fortran 77.

We have spent a lot of time in the last five years scratching our heads over C++ (and over Java in the last couple of years). Probably a "Numerical Recipes in C++" would have value. There are several reasons, however, that we have not produced such a version.

First, the original "democratic" dream of object-oriented programming, that every programmer would accumulate an idiosyncratic collection of useful object classes whose reusability would allow moving to ever higher levels of programming abstraction in the course of a career — this dream seems dead. Instead, today's trend is toward a fixed, universal object-class library (Microsoft's MFC, more or less) and is discouraging of more than a miminal amount of idiosyncratic programming at the object-class-definition level. The result is that C++ has become essentially a large, but fixed, programming language, very much oriented toward programming Microsoft Windows software. (Fortran 90, on the other hand, is strongly biased toward scientific computing — it is a poor language in which to write native Window's applications!)

ISSN 1058-9244 / $8.00 © 1999, IOS Press. All rights reserved

Second, there is a genuinely unresolved debate regarding what should be the fundamental structure of a scientific programming library in C++. Should it be true to the language's object-oriented philosophy that makes methods (that is, algorithms) subsidiary to the data structure that they act on? If so, then there is a danger of ending up with a large number of classes for highly specific data structures and types, quite complicated and difficult to learn, but really all just wrappers for a set of methods that can act on multiple data types or structures. There exist some ambitious class libraries for scientific computing (see, for example, Ref. 14) that suffer, to one extent or another, from this problem.

Confronting just this issue, a competing viewpoint, called "generic programming with the Standard Template Library (STL)" has emerged. Here algorithms and data structures ("containers") have more-equal claims to primacy, with the two being connected by "iterators" that tell the algorithm how to extract data from the container. STL is implemented as C++ template classes that naturally allow for multiple data types (for example, single versus double precision).

We do not feel ready to choose one of these C++ methodologies, and we have only just begun thinking about what we might conceivably propose as an alternative. One possibility would be to define a generic, very-high-level, interface that encapsulates a set of objects and methods comparable to everything in Fortran 90, but not in itself dictating any particular template or class-inheritance structure for its implementation. Then, a variety of compatible implementations could be written, optimized quite differently for today's serial or tomorrow's parallel machines. Our preliminary efforts along these lines are at http://nr.harvard.edu/nr.cpp, and we would be grateful for thoughts and comments from our readers.