

Title: A Sneak Peek at New Fortran Standards – *Should finance house quants really care?*

Summary: Some of the more prominent quantitative analysts working for international banking institutions are strictly Fortran users now and have been for quite some time. There are compelling reasons why Fortran is likely to grow in relevance in the coming years, especially in light of the new standards being finalized by both the U.S. and international standards committees charged with formalizing Fortran 2003. Evolving Fortran standards also underline the reality that multi-language applications are the new norm, and for good reasons.

The presumption in many, if not most, finance circles is that C/C++ is *the* programming language, especially whenever the computational limitations of JAVA get in the way. That is not as universally true as one might think. The Numerical Algorithms Group (NAG), for example, creates mathematical, statistical and three dimensional visualization software components in all major languages and is quite agnostic as to which one users prefer. Last year when NAG introduced its first Fortran compiler for the Mac OS X platform, we were a bit surprised that it quickly became NAG's fastest selling compiler. In fact, some of the more prominent quantitative analysts working for international banking institutions who use NAG code to build their applications are strictly Fortran users now and have been for quite some time.

Moreover, there are compelling reasons why Fortran is likely to grow in relevance in the coming years, especially in light of the new standards being finalized by both the U.S. and international standards committees charged with formalizing Fortran 2003. Evolving Fortran standards also underline the reality that multi-language applications are the new norm, and for good reasons.

The superiority of JAVA front ends for many commercial applications is widely recognized. Relatively easy-to-access JAVA front ends bring relatively non-technical users seamlessly into a computing engine. However, the three-to-10 time performance hit one gets in numerical computing in JAVA makes it a poor fit for any numerically intensive computational task. An ideal application might get data in via JAVA or an equivalent language, but if you want raw computational speed, C and Fortran are the superior options.

In many respects, and certainly when compared to JAVA and the like for the types of computational tasks quantitative analysts were handling a few years ago, C and Fortran are comparable when it comes to computational speed. However, from a numerical point of view, Fortran is inherently safer for certain operations, compared to C. For example, C allows aliasing, which makes it more difficult for a compiler to optimize code. Also, unlike Fortran, C has poor multidimensional array support. The C99 standard has tried to address some of these limitations but these features are not yet widely available.

The modifications to Fortran, pending with Fortran 2003, build upon the changes initiated in prior Fortran releases to make the language work better with other languages in the growing number of multi-language operations. On the one hand, this reflects the strong desire by many to enable Fortran to endure and on the other hand to make sure that support for other languages is built into Fortran properly.

The endurance of Fortran can be attributed to several factors. First, Fortran's performance in numerical computing is and has always been the key ingredient to its usefulness. Second, from the perspective of many Fortran users, especially in scientific fields, Fortran can not only produce highly reliable code, but is easier to write and easier to maintain. Unlike C, much of Fortran has in fact been written by physicists and engineers, as it is designed to be friendly for such non-programmers to write and also to go back to it later and understand what has been written. In comparison, good C programming is much more dependent on a talented programmer, and C++ is even more difficult for the novice to write maintainable code in. Third, there are many computational problems that are best solved using arrays, for which

Fortran is especially well-suited as compared to C, C++, or JAVA. The specialist array languages have all died or are hopelessly slow, leaving Fortran as the premier language for array types of computation. And lastly, and perhaps most importantly from a finance perspective, the massive amounts of data now being handled, compared to earlier times, and the growing importance of simulation exercises, makes it clear to those of us that are more language-neutral in disposition, that Fortran is often a better option.

When one is trying to do computations with large datasets of stock market tick data, such as when applying time-series models to optimize projected returns, or whether one is handling very little data but then generating enormous datasets with random number generators or comparable simulation techniques, the scale of quantitative analysis in the finance arena is now on the level of many giga-bytes of data per run and often array processing takes on more significance. If an array processing type approach better matches the computational problem than a pure object oriented programming approach, Fortran would be the better choice and vice versa. The massive size of datasets is making performance an issue more often, and this in turn goes back to using computational approaches that do not particularly match the type of problem being handled.

Fortran was born of the need to find robust methods to cope with such huge amounts of data, and performance in those situations has been one of the driving factors throughout the evolution of the language.

In contrast, if one is not doing simulations and the dominant part of a problem requires low-level system interaction (for example Posix or Win32), a GUI user interface, and is other than arrays of numerical data, C is a better language choice. C's popularity is in part a reflection of it being the language that most post-baby-boomers learned in college and in part the result of Unix being very C-oriented. In Unix systems, all the systems calls are written in C and many other software packages such as GUIs have a C interface without provisions for any other languages. If you want to use it, you have to write in C. Depending on the relative amounts of code, that can be a good decision or a bad one.

C and Fortran both have reasonable string handling, although strings work differently in each language. Fortran strings are perhaps slightly less flexible than C strings but are more resistant to the dreaded "buffer overflow".

Given all these pros and cons of various languages, Fortran and other languages are developing in ways that accommodate the trend towards multi-language applications. Although standards committees purposely move glacially slow so as to ensure that standards changes are enduring, the recent changes in Fortran are likely to make it more accessible to those with less Fortran orientation such as quantitative analysts and others dealing with newly huge datasets.

As in earlier versions, Fortran 2003 continues to add features that will help it better mesh with more recent developments in programming paradigms. The revisions in Fortran 2003 have been quite thorough. They range from the trivial (longer names up to 63 characters, more continuation lines, longer statements) to the more substantial additions to support object-oriented programming and ensure that Fortran is consistent with the IEEE arithmetic and C language standards. The new Fortran has many features that make it easier to join C programs and Fortran programs together including support for reading and writing files written in C that are read in Fortran as stream files. Most C APIs can be described and directly called from Fortran 2003. The object-oriented features are based on type extension, polymorphic variables, and type-bound-procedures. This fits the pre-existing Fortran language better than the more usual class-oriented model, while still providing inheritance and object methods (dynamic dispatch) in the expected manner.

On the other hand, in the new Fortran 2003 standard development, much attention was paid to not go overboard in revamping Fortran into an object-oriented approach. In fact, C++ was a bit of a model as to

what not to do, such as in using multiple inheritance techniques that adversely impact performance. Similarly, making everything a polymorphic object was avoided as this too would take away from Fortran's outstanding ability to handle the massive datasets and its unique array processing approach.

Will quantitative analysts be affected by this new Fortran standard?

In the short term, language-neutral vendors such as the Numerical Algorithms Group expect that most people, in all fields, will continue to use whatever language they know best because it is easier for them to do so. Currently, new Fortran compilers come out in the marketplace on a 14- to 18-month cycle, reflecting the healthy and enduring demand for that language's superior attributes. To the extent that quantitative analysts seek better performance with growing datasets, Fortran may no longer be one of those best-kept secrets of a few select finance houses, but become more widely recognized as a powerhouse for numerical computing that has its place in multi-language applications.

By Malcolm Cohen, Principal Technical Consultant of the Numerical Algorithms Group. Malcolm has worked with both the U.S. and International Fortran Standards Committees for 15 years.

Originally published by Financial Engineering News, January/February, 2004 (www.fenews.com)

Numerical Algorithms Group

www.nag.com / info@nag.com (*North America*)

www.nag.co.uk / info@nag.co.uk (*Elsewhere*)