N362
94-047

Subject: Comments on complex vs imaginary types for C

I wish to strongly support the introduction into the programming language C of a type imaginary, rather than a Fortran-like type complex. The essence of my argument is that experience with Fortran has shown that the representation of complex numbers by a pair of adjacent floating point values is too restrictive, and consequently in areas such as signal processing where substantial computation with complex arithmetic arithmetic is done, the complex data type in Fortran is almost never used - the arithmetic is simulated with reals, as would be done in languages such as Ada or current C which do not directly support complex arithmetic. Yet appropriate language support, i.e. type imaginary, would make programs easier to produce, would make them easier to read, and arguably might make generation of better code possible. The examples below all have difficulty with Fortran-like representation, and all have no difficulties when real and imaginary are separate types.

There are basically two problems with the Fortran representation. First, it is not always convenient that the two floating point numbers making up the real and imaginary parts lie at adjacent addresses in memory, Second, although abstractly values may be complex, in fact it is very often desirable to take advantage of the fact that in special cases these values may be provably real, or imaginary, or exist in complex conjugate pairs, etc. Not only does this facilitate saving storage space and computation time, but it may be essential to preserving properties that might be accidentally lost by roundoff or algorithmic approximation. Preserving and exploiting these special cases typically involves treating the real and imaginary parts of a complex value separately. This is awkward to do with selectors accessing the parts of complex number.

As a simple example where storing an array of complex numbers, represented as pairs, is not convenient consider a Hermitian matrix. This kind of matrix, arising commonly in physics, has the property that it is its own Hermitian transpose, ie that the elements of the transpose of the matrix are the complex conjugates of the elements of the original matrix. Only half of the off diagonals thus need to be represented. Moreover, the diagonal must be real, thus making it possible to save the N locations that would be occupied by the imaginary parts that are known to be zero. If the redundant parts are explicitly represented, and if the Hermitian transpose property is accidentally lost, say through roundoff, then the related property that the matrix is positive definite, i.e. that the eigenvalues are real and indeed positive, will also be lost.

A more significant example concerns the finite Fourier transform of a sequence of real data. This transform has the property that it is Hermitian symmetric, i.e. that the Fourier coefficient f(t) is the complex conjugate of the Fourier coefficient f(N-t). This implies f(0) is real, and if the sequence is of finite

length N which is even, then f(N/2) is real. Starting from a real sequence of N values, the independent information computed by the Fourier transform are the N values f(0); Re(f(k)),k=1..(N-1)/2, Im(f(k)),k=1..(N-1)/2, and f(N/2) if N is even. Conseqently the amount of storage is conserved: the transform can be done in place. If, however, f(0) and f(N/2) must be represented as complex number pairs, then the storage requirement increases. The practical importance of this can be seen when the Cooley version of the FFT is applied to a sequence of real data. At each stage, the Cooley version of the FFT can be described as combining the Fourier transforms of subsequences of the original sequence to give the Fourier transform of the combined subsequence; eventually this leads to the Fourier transform of the original sequence. Each subsequence is of course real, hence each intermediate Fourier transform is Hermitian symmetric and can be done in place if no redundant storage is used, ie if only the independent information noted above is stored. The subscript values required for indexing are simplest if imaginary parts Im(f(k)) are stored at location N-k in the array; in particular the subscript values required if an vector of complex values is used and redundant storage is avoided is much more complicated that "bit-reversed" indexing conventionally used with FFT's.

Despite all the work since that time, (including the advent of the language C), little has changed since these issues were discussed in 1970 at the Fourth Annual Princeton Conference on Information Sciences and Systems in a session on the Use of Complex Arithmetic in Numerical Analysis. Papers included:

Jenkins, M.A. "The advantages and Disadvantages in Using Complex Arithmetic in Polynomial Zerofinding", pp129-132.
Businger, P.A. "Using and Avoiding Complex Arithmetic in Linear Algebra", pp133-135.
Sande,G. "Fast Fourier Transform - a Globally Complex Algorithm with Locally Real Implementations", pp 136-142.
Kahan, W. "Where does Laguerre's Method come from?", pp143-145.

Some quotes are appropriate:
"It is quite natural to consider the solution of polynomial equations as a problem in the complex field and to program algorithms entirely in complex arithmetic. However, most polynomials solved in practise have real co-efficients and there exists efficient algorithms for real polynomials which use no complex arithmetic.
... A comparison is made which shows that the algorithm for real polynomials is considerably more efficient than the algorithms for complex polynomials. However, the complex algorithm is simpler, easier to analyze and somewhat more reliable." (Jenkins)
"The issue of whether to use or avoid complex arithmetic in linear algebra arises with problems whose data or solutions are complex. While it is possible

to avoid complex arithmetic even when data as well as solutions are complex, using complex arithmetic in such cases is more economical. An instance of complex data leading to real solutions is given by the Hermitian eigenvalue problem. In the interests of efficiency the computation can be arranged such that a significant part of the work is carried out in real arithmetic. In the case of the real nonsymmetric eigenvalue problem, real data in general lead to complex solutions. Avoiding complex arithmetic in this case also leads to a gain in efficiency."(Businger)
"We may now ask why [good] Fourier transform programs do not use complex types. At the arithmetic level the answer involves the statement 'multiplying by i is not efficient'. We also find other special constants and combinations of interest. Good Fourier transform programmes are a thorough study in exploiting the special properties of the complex exponential....Thus we find that, locally, complex arithmetic is expensive and that, globally, complex data types are inappropriate." (Sande)
"What lessons have we learned which we might apply in the future when specifying new languages. Complex constants may be of special form and this should be acknowledged. This is very important in this algorithm and is important in many other situations. There is a difference between the notions of complex array and array of complexes. For this algorithm it matters crucially." (Sande)

Extensions of some of the ideas in Sande's paper to related problems, as well as other similar techniques, appear in:

W.M. Gentleman, "Using the Finite Fourier Transform", Computer Aided Engineering, Study No. 5, (edited by G.M.L. Gladwell) Solid Mechanics Division, University of Waterloo, Waterloo, Ontario, pp. 189-205, 1971.
W.M. Gentleman, "Implementing Clenshaw-Curtis Quadrature II, Computing the Cosine Transform", Communications of the ACM, Vol. 15, No. 5, May 1972, pp. 343-346.
W.M. Gentleman, "Algorithm 424, CCQUAD - Clenshaw-Curtis Quadrature", Communications of the ACM, Vol. 15, No. 5, May 1972, pp. 353-355.

Date: 8 Jun 1994 18:20:10 U
From: "Gentleman" <Gentleman@iit.nrc.ca>
Subject: Comments on complex vs imag
To: "Jim Thomas" <jim_thomas@taligent.com>