WHITHER (WITHER?) CONTROL STRUCTURES?

by Clark Wiedmann
APL Language Editor

The following paper, "Special Control Structures for APL" by A. P. Reeves and J. Besemer, is one of several that have proposed extensions of APL to provide control structures. As Language Editor of APL Quote Quad, I have endorsed publication of the paper because Quote Quad has not served recently as a forum for discussing control structures, and because the paper contains some ideas that may be improvements over past proposals. However, I cannot with good conscience let the paper pass without noting that the more fundamental question of whether control structures improve the language still has not been answered.

On the surface, it appears that control structures would simplify programming. Many of us have yearned for a simple, foolproof way to perform some statements *N* times (where *N* may be 0), without needing to initialize counters, increment counters, and compare counters, and without involving four statements and two labels in the task. But it is clear that control structures would add some new elements and rules to the language. The question is whether the added complexity of the language is justified by improved clarity and reliability of programs.

A few years ago, control structures experienced a great revival, and it appeared to many that APL could not survive without them. At that time, APL was roundly denounced at gatherings of computer scientists for its lack of control structures, and many APL implementers looked favorably on adding control structures to the language. But, time has passed, and the control structure imperative has faded. APL has survived, and many of us are still eight times more productive when programming in APL than in the structured languages. The computing community seems to have realized that there is more to the art of programming than merely using control structures, and increased respect for APL has emerged among computer scientists. In 1976, few APL implementers still favored adding conventional control structures to the language. Instead, many favored further exploration of alternative mechanisms such as new data structures (general arrays), operators, functions, and event control. It remains to be seen if the rest of the APL community feels that control structures are unimportant. Unless a need can be demonstrated, I think further discussion of what structures are best will be pointless.

Experience with other languages may not be relevant to APL; it has been observed that control structures sometimes divert APL programmers from more powerful tools. In present APL the use of arrays and array-oriented functions has eliminated much of the need for control structures. That is, structure in our data has taken the place of control structures in our programs. Future extensions to the language can be expected to further lessen the utility of control structures. The danger seems very real that programmers would use control structures rather than finding better solutions that involve arrays. For example, one eminent member of the APL community gave APLGOL an honest try, but he was shocked to discover how carried away he became -- he found he had used nested loops to perform a matrix product rather than using +.×.

As an example of improving a program without the use of control structures, consider the function *M*2 which has been used in some papers to show the need for control structures. The function first appeared in a paper by Woodrum [1], then in papers or memos by Jenkins [2], Orgass [3], Foster [4], and now Reeves and Besemer. The version shown below was simplified by adopting slightly different data structures. The original program (see the end of the following paper) had 13 statements (7 branches), while this version has 4 statements (1 branch). Incidentally, the version below corrects two errors that have been propagated since the Jenkins paper -- the more recent algorithms would not actually work. Also, this version performs the same number of comparisons as the original, which is of some importance since the original algorithm demonstrated sorting with minimal comparing. The interface has been changed in the version below. Rather than returning an explicit result, the first link of the result chain is stored in the last element of *P* (an extra element to be provided by the caller). Also, rather than using two scalar arguments *I* and *J*, a vector argument *IJ* is used. As in Woodrum's paper [1], zero origin has been assumed.

```
    ∇ M2 IJ;T;W
[1]   T←ρA
[2]  L1:T←P[T]←IJ[W←≥/A[IJ]]
[3]   →(T≠IJ[W]←P[T])/L1
[4]   P[T]←IJ[~W]
    ∇
```

References

1. Woodrum, L. J. Internal sorting with minimal comparing. IBM Systems Journal 8, 3, (1969).

2. Jenkins, M. A. A control structure extension to APL. Report No. 21, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1973.

3. Orgass, R. Structured *DEC*10*APL* user's manual. Technical Report No. APLAD3, Department of Computer Science, University of Arizona, Tucson, June 1975, revised 11 July 1975.

4. Foster, G. A. What lies beyond the branch arrow? APL 75 Congress Proc. (Pisa), ACM, New York, 1975, pp. 115-122.