Introduction

Enhancing Forth

Forth is a singular language. It is used where other languages tend not to fit, such as in compact, ROM-able systems, or in highly interactive applications. Partly because of its lack of competition, its adherents have tended to ignore features and structures of other languages. Even so, most Forth programmers recognize that there is room for improvement in Forth. Indeed, it has been observed that the first thing a Forth programmer does is enhance his/her system by customizing it to the programmer and application at hand.

The introduction of an individual system's features to the entire Forth community is a slow process. For example, most Forths have a multitasking capability via user variables or state vectors, but many programmers don't use multitasking, even after ten years of availability. Other enhancements, like Bartholdi's TO [BART79] and Rosen's QUAN [ROSE82] for data structures, or Ragsdale's ONLY [RAGS82] for vocabulary search order, have been widely disseminated, and are now being put to use. On the other hand, Shaw's vocabulary stacks [SHAW82], Kitt Peak's orphan and pseudo vectors [BART77] and Schleisiek's number input formatting [SCHL82] were never widely adopted.

This issue of the *Journal* contains several papers concerned with enhancing the Forth environment. Duff and Iverson's paper, "Forth Meets Smalltalk" describes a Forth-like language, Neon, for the Apple Macintosh computer. They took their cues from the Smalltalk community, but with one exception: performance. Smalltalk's contribution to Forth is the idea of object-oriented programming, or the definition of classes of objects with embedded rules for their manipulation, and the definition of instances of a particular class of objects. These are constructed in Forth using defining words which define defining words [FORS82]. Neon further enhances Forth by adding named input and output parameters, and local variables, via a third, methods stack. Duff and Iverson maintain Forth's speed as shown in their benchmark (Figure 11).

Similarly, Korteweg and Nieuwenhuÿzen address local storage in their paper, "Stack Usage and Parameter Passing". Like Neon, their system allows for named local variables via an additional stack. They discuss implementation tradeoffs and run-time parameter code techniques. This work was preceded by Bartholdi's paper on parameters [BART82]. Both the Duff and Korteweg papers reflect a trend towards higher level operators and away from low level stack and data storage manipulations: indicating that *what* is being done is more important than *how* it is done. It is an exercise for the reader to determine whether these concepts enhance Forth, or supercede it.

MacIntyre addresses the subject of floating point calculations in his "Number Crunching with 8087 FQUANs: The Mie Equations". MacIntyre solves a series of functions which relate a small bubble's diameter to its light scattering spectrum. He notes that highly iterative computations with wildly varying dynamic ranges can't be solved using Forth's integer scaling operator */ or its extended brethren. Although his code took but a second of CRAY computer CPU time, it required 24 hours of turnaround; whereas he was able to compute the same equation in 15 minutes on an IBM PC with an Intel 8087 floating point co-processor.

The subject of Michaloski's "A Forth Profile Management System" is the problem of source code control in large projects, one of the recognized problems in Forth. Although the manipulation of individual screens rather than files is efficient in testing, it often becomes unwieldy for large programs. Michaloski describes a compact package using commenting structures to break up screens, and parts of screens, into independently loadable and trackable partitions, or profiles. His system is analogous to the use of : definitions to define new relationships. Only here, the relationships evolve from testing and maintenance needs. The author uses the underlying Forth block structure to implement a system more useful than a simple, or even hierarchical, file system. As he notes, ". . . the profile management system is only a stepwise upgrade from a Forth block management system".

The technical note by Ko and Jenson on "Patching the ONLY Structure on Top of fig-FORTH" and an algorithm by Duff on "A Group Construct for Field Words", continue this issue's theme. Ko's technical note includes 10 screens of source code indicating how to upgrade the limited FIG vocabulary structure to that proposed by Ragsdale [RAGS82]. Duff's algorithm also deals with data structures, and improves upon the work of Haydon [HAYD81] and others by providing a replication mechanism for aggregations of fields in repeating groups.

It is no coincidence that each of these papers builds on ideas developed over the past few years by work in Forth and other fields. Just as these papers grew, they may inspire others to further enhance Forth. It will be interesting to see where this issue of the *Journal* leads over the next few years.

Lawrence P. Forsley
University of Rochester

References

[BART77] Bartholdi, Paul, "Pseudo-Vectors EFUG Notes", *European Forth User's Group*. June, 1977.

[BART79] Bartholdi, Paul, "The 'TO' Solution", *Forth Dimensions* Vol.I, No 5. pp 38–40. January/February 1979.

[BART82] Bartholdi, Paul, "Another Aid for Stack Manipulation and Parameter Passing in Forth", *1982 Rochester Forth Conference Proceedings* pp. 221–226.

[FORS82] Forsley, Lawrence P. "Recursive Data Structures", *1982 FORML Proceedings* pp. 205–207.

[HAYD81] Haydon, Glen, "Elements of a Forth Data Base Design", *1981 Rochester Forth Conference Proceedings* pp. 165–177.

[RAGS82] Ragsdale, William F. "The ONLY Concept for Vocabularies", *1982 FORML Proceedings* pp. 109–115.

[ROSE82] Rosen, Evan, "High Speed, Low Memory Consumption Structures", *1982 FORML Proceedings* pp. 191–196.

[SCHL82] Schleisiek, Klaus "NUMBER Input Wordset", *1982 FORML Conference Proceedings* pp. 147–153.

[SHAW81] Shaw, George W. II, "Executable Vocabulary Stack", *1981 FORML Conference Proceedings* pp. 117–120.