Introduction

Using and Improving Forth

This issue of the *Journal* covers both the traditional use of Forth, in real-time applications, and extensions to Forth inspired by other programming languages. The first two papers describe real-time programming systems, one for ultrasonic imaging, and the second describing a flexible, indeed almost chameleon-like, system for acquiring data in a femtosecond laboratory. The next three papers, and a technical note, discuss various ways of improving the readability and maintainability of Forth code through a variety of mechanisms including local variables, named parameter passing, compile-time binding for Bartholdi's VALUEs and an exception handler.

Lynk and Johnson's paper, "Forth-Based Software for Real-Time Control of a Mechanically-Scanned Ultrasonic Imaging System" describes both an experimental system and techniques useful in real-time programming. These techniques include a high-level interrupt handler and a memory management mechanism specific to the Digital Equipment Corporation PDP 11 series computer. One stunning result from the system is an ultrasonic picture taken *through* a copper penny!

Similarly, Sizer's paper, "Fast and Flexible Forth Programming in a Femtosecond Laser Lab" discusses another real-time application: the need to provide a flexible programming language for a variety of short pulse laser-based experiments. In addition, Sizer has had to emulate instruments like box car integrators, signal averagers and discriminating photon counters. He takes considerable advantage of vectored execution in both organizing experiments and in controlling similar, but different, devices like stepper motors.

Lyons' paper, "Stack Frames and Local Variables" reviews existing work in this area in Forth, which is largely borrowed from Algol-like languages. He observes that the major difficulty with stack manipulation is less that of naming elements on the stack and more one of easily accessing beyond the top three elements. Lyons suggests that although several mechanisms have been proposed or implemented, a hardware indexed parameter stack is the appropriate place for a stack frame.

Lewis' paper, "Should VARIABLE Be an Immediate State-Sensitive Word?" also reviews previous work: this time Bartholdi's TO, Rosen's QUAN and Schleisiek's Multiple CFAs, before proposing an alternative. He believes that prefix operators, as first proposed by Bartholdi, are useful, but slower than necessary. Similar structures proposed by Rosen and Schleisiek are faster, but are non-standard. Hence, Lewis develops a state-smart version of TO, with extensions, which is compatible with the Forth-83 standard.

Like Lyons and Lewis, Stoddart appreciates the usefulness of local variables and standardization in his paper, "Readable and Efficient Parameter Access Via Argument Records". Similar to Lewis, he proposes a prefix operator with the Smalltalk-like notion of an access method for operating on a specific instance of a word, but extends this to encompass Lyons' stack frames. He also presents an implementation in Forth-83 compatible code.

The technical note by Paul, et al., "Run-Time Error Handling in Forth Using SETJMP and LNGJMP for Execution Control" also borrows a useful technique from outside Forth, this time the UNIX/C SETJMP and LNGJMP. They observe that implementing a run-time error handler in Forth is often cumbersome, and have developed Forth code for a controlled GOTO. Their technical note concludes with the code for implementing two versions of the technique.

It is significant that two of these authors, Lewis and Stoddart, have found that their enhancements can be conveniently made within the context of the Forth-83 standard. Indeed, Stoddart comments "...I have been surprised how effectively Forth can describe the enhancements and integrate them into its repertoire." Thus, Forth continues to have room for improvement, and room for that improvement.

Lawrence P. Forsley
University of Rochester