Introduction

Application Languages

This issue of the *Journal* contains three papers describing Forth enhancements that are effectively application languages, and a fourth which provides dynamic storage management capabilities useful in these and many other contexts.

Cotterman *et al*'s paper, "A Microcoded Machine Simulator and Microcode Assembler in a FORTH Environment" describes a set of tools used in graduate courses in computer architecture at Wright State University. This work is similar to that of Cholmondeley[1], who published a microcode assembler, and Wyckoff[2] who wrote a microcode simulator. However, the group at Wright State has gone much further in incorporating both a general-purpose simulator and an assembler, and then using them for instruction and computer design. As an example they present a hardware representation of the venerable Digital Equipment Corporation PDP-8.

The paper by Dress, "A Forth Implementation of the Heap Data Structure for Memory Management" presents code for dynamically managing memory. His work was inspired by observations of MacFORTH's[3] access to the Macintosh ROM-based heap manager. Although Dress originally developed it to support the OPS/5 expert system programming language, the heap is useful for many applications, including one suggested by Pountain in this issue.

The *Journal*'s second foray into aspects of Smalltalk can be found in Pountain's paper, "Object Oriented Extensions to Forth". In contrast to Neon, described by Duff and Iverson[4], Pountain took a simpler approach, allowing him to publish the complete source code. Object oriented programming offers the promise of both more reliable and simpler code; and implementations in Forth may sufficently reduce the speed and space penalties of this methodology so as to allow for real world, and possibly real-time, applications using it.

Forth's finesse in efficiently implementing a variety of application languages continues with Zettel's paper, "Discrete Event Simulation". His simulation deals with the problem of waiting lines, or queues, and he uses a classic simulation problem: queueing up in a barbershop. Zettel maintains an event calender as a linear linked list in the dictionary, and notes that the Forth text interpreter allows interactive interaction with the simulation – a capability which he hasn't seen in other simulation systems.

We conclude the papers with a technical note by Miller and Dowling, "Plotter Drivers as an Exercise in Forth Wordset Design" where they present the source code for a driver and give examples using the Radio Shack FP-215 plotter. The author's observe three levels of sophistication in developing their plotter application language: mimicing the plotter's capabilities, developing an English-like wordset and incorporating the plotter into a standardized graphics language.

Although Forth may seem to lack the "computer graces", it is sufficiently rich in form and concise enough in structure to allow us to publish the complete source code for all but one of these papers. This richness allows us to build languages which are closer, in syntax and semantics, to the applications we are trying to implement. Whether simulating a computer or a barbershop, implementing objects, heaps, or turtle graphics, Forth provides a profound set of tools to build tools.

Lawrence P. Forsley
University of Rochester

1. Cholmondeley, G.,"A FORTH Based Micro-Sized Micro Assembler", *Forth Dimensions*, Vol. III, No. 4.
2. Wyckoff, R.,"A FORTH Simulator for the TMS 320 IC", *1983 Rochester Forth Conference.*
3. Creative Solutions, Inc., *Multi-FORTH Version 2.00 User's Manual.*
4. Duff, C. and Iverson, N., "Forth Meets Smalltalk" *Journal of Forth Application and Research*, Vol. 2, No 3.