Introduction

In the 12 years or so that I have been using Forth, I continue to be amazed at the power provided by its inherent extensibility. Many programming languages provide much richer tool sets, but I've never worked with a language with such a simple means of constructing the exact tool set required to solve a given problem. This issue of the Journal examines some issues associated with extensibility along with some useful extensions.

I believe the tone for the issue is set by a very thoughtful letter from Brad Rodriguez which focuses on some key ideas about using Forth. In particular, Brad echoes two sentiments that I've heard often enough in the Forth community, but which still seem to be as relevant as when they were first raised. Forth is perfect for building small embedded systems yet the tool set provided with most environments is woefully inadequate. While I know one can write any tool one needs with Forth rather easily, issues today in software engineering revolve more than ever about features such as reusability, reliability, and maintainability. To succeed in this arena will require us as a community to address how Forth can better provide such features.

The Cost of User-Friendly Programming by Dr. Ferren MacIntyre *et al.* gives insights into just where Forth's extensibility can make a real difference in the overall applicability of a software system. The authors observe that while the tendency toward slick user-friendly interfaces have made software easier to use, it has also taken its toll with respect to the ability to tailor a system. In general, such interfaces result in increased code and decreased programming options. Implementations with Forth can create easy to use software that can still leave room for the user to optimize code for specific applications.

Dr. Karl-Dietrich Neubert delineates one methodology for creating objects in Forth in *Little Universe: a Self-Referencing State Table.* By using a state table as the underlying structure, Neubert creates a flexible environment in which one can embed a variety of actions or procedures. The paper contains code for the basic implementation as well as examples illustrating its use in applications such as knowledge bases and active systems.

One of the most difficult aspects of learning Forth for the traditional programmer is often effective use of the stack, especially for intense mathematical calculations. *A FORmula TRANslator for Forth* by Dr. Julian Noble explains the ins and outs of implementing an expression parser in Forth and provides code for a rather complete formula translator. While not every Forth application requires this kind of extension, it proves useful for scientific programming in Forth.

Exception handling in a production environment is a crucial element of a language's ability to construct reliable applications. Ms. Carol Pruitt's *A Generalized EXIT* summarizes a flexible yet simple way to implement a means by which a Forth word can terminate its own execution upon discovery of an exception condition. This further enhances Forth's ability to handle exceptions in a uniform way.

The final paper in this issue, *Strings, Associative Access, and Memory Allocation* by Dr. Nicholas Solntseff and Mr. Bradford Rodriguez, explains the underpinnings for implementing a flexible string structure and associated operators in Forth. For many years, languages such as SNOBOL and BASIC have been lauded for their ability to handle operations on character strings. Some Forth systems provide some string capabilities, but are often lacking in most implementations. This paper forms a base point for a comprehensive tool set.

Its always encouraging to be able to provide new techniques and implementations. That always has been and always will be part of the excitement of Forth programming. Yet, I hope that as a community we can begin to get a handle on providing more access to basic tools to solve problems using Forth. In this age of super languages like Ada and C and Modula-2, Forth has to begin to offer more tools as well as excitement to be an effective contender. How about your reactions?

James Basile
Editor-in-Chief, JFAR