Introduction

This second issue of *The Journal of Forth Application and Research* is built upon the theme of data structures and their implementation in the Forth environment. It is well recognized that having a clear conceptualization of the relationship among data items is essential for formulating effective, efficient solutions to a problem. In most programming environments, however, the programmer finds a challenge in the task of implementing such relationships concretely.

Forth has long been touted for its extensibility, yet the power of this concept has often been neglected relative to data structures. Forth provides all the capabilities that PASCAL does for user defined data types and structures, and then goes further. The CREATE ... DOES> construct allows the programmer not only to define unique structures, but also to attach action to the structure. As part of a total environment, these capabilities are awesome.

This issue examines some practical applications of user defined data structures in Forth and, hopefully, serves to kindle continued discussion in this important area. Specific uses of such structures to accomplish design goals are recounted.

The first paper, *Implementing Data Structures in FORTH,* is a detailed tutorial on how to use Forth to create varied user data structures. It relates how to implement abstract structures via the CREATE ... DOES> construct, but also explores uses beyond simple data structures.

Vectoring Arrays of Structures emphasizes how useful Forth can be in creating modular code. By using special pointer variables to tables of actions, it is possible to isolate the physical characteristics of device access from the logical characteristics. The techniques developed apply to a broad range of applications.

Another important application technique is described in *Message Passing with Queues.* In a multi-tasking and/or multi-user environment there is a need for passing information among tasks. This paper details the solution for this problem implemented on the Omega Laser Alignment System at the University of Rochester.

A theoretical treatment of the use of "intelligent" data structures is presented in *Multiple Code Field Data Types and Prefix Operators.* Expansion of the "TO" concept via use of certain prefix operators is proposed.

The final paper, *A Forth Random Number Generator,* is a concrete description of how to implement an efficient yet good random number generator in Forth. 79-Standard code for the implementation is provided.

This issue marks the inauguration of a new section for *The Journal.* While Forth is so extensible, one often hears the complaint, "You can do anything you want in Forth, only you have to develop all of it from the basics." *The Journal* will now contain an ongoing section called "Extensions" which will provide a catalogued forum for exchanging structures and algorithms. If you have a structure or algorithm which you think has general use and which you want to share, please consider submitting it for consideration in this section. We all stand to benefit from the exchange of information. Code for the routine should be accompanied by a brief (one page) description of how it is used and any deviation from standard code.

I want to thank all of you who have worked so tirelessly to contribute your ideas and efforts to this issue, the authors, the reviewers, and especially those who worked behind the scenes putting it all together. May the second year of *The Journal* see continued strides made toward providing a beacon for the Forth community.

James D. Basile
Long Island University