The Need for a Standard Test-Suite

Nicholas Spies
313 Grace Street
Pittsburgh, PA 15211-1503

Mahlon G. Kelly,
268 Turkey Ridge Rd.,
Charlottesville, VA 22901

A test-suite is a set of programs used to assure that a compiler or programming language meets explicit specifications. A standard FORTH test-suite would bring many advantages to the FORTH community. Ambiguities in the language of the standard would be resolved through example. Both users and system-authors would have a convenient way to confirm that their dialects meet standard requirements. And most importantly, dialects that could run the standard test-suite successfully would be virtually assured of being code-compatible. In short, we feel that a standard test-suite is a necessary step in the evolution of an effective FORTH standard.

But before a standard FORTH test-suite is written some questions must be answered. These include: How would a FORTH test-suite be designed and written? How reliable, redundant or inclusive would it be, or would it have to be? In what dialect or dialects would it be developed? Who would write it, and by what process would it be approved by the standards team? Would a standard test-suite be accepted by the FORTH community? Would the effort be justified? Of course many of these issues will have to be resolved by discussion and consensus so the discussion that follows will be restricted to some thoughts on how a standard test-suite might be designed.

Each FORTH word may be considered to be a "black box" whose stack input, output, and actions are defined in the standards but whose implementation is left to the authors of standard FORTH systems. The test-suite would test each word for its proper function with particular attention being paid to the allowed limits of its arguments and for interactions with other words and memory as appropriate. A complete test-suite would need to perform an undetermined number of redundant tests to assure that all words meet standard requirements. The material presented here raises more questions than it answers and is meant to stimulate thought about a test-suite rather than to offer a particular solution. The standards team would have to determine the form and scope of the test-suite.

The first step in defining a test-suite would be to give examples for each definition in the required word set so as to make explicit the various conditions and limits of the use of the word. For example the FORTH-83 definition of + is simply

```
    +       w1 w2 -- w3                 79              "plus"
         w3 is the arithmetic sum of w1 plus w2.
```

but the definition is almost circular in that "arithmetic sum" is practically synonymous with the outcome of the operation "plus". The definition should address the question of how + transforms w1 and w2 to give w3. Giving some examples in the definition would leave less to guesswork or prior knowledge. If some limiting conditions such as

```
 32767  32767 + U.  displays  65534
-32768 -32768 + U.  displays      0
-32768  65535 + U.  displays  32767
```

were defined the action of + would be made more explicit. These kinds of examples, even for "self-evident" definitions, would go a long way toward making the standard more informative (and would probably force the re-writing of some vague definitions). Of course the standards team would have to decide what examples would be appropriate.

Once the limiting conditions are determined they could be used to define a simple word to check whether + functions properly. This might be done with

```
: +TESTER  ( -- )  32767  32767 + 65534 =
                  -32768 -32768 +     0 = AND
                  -32768  65535 + 32767 = AND
  IF ." + passed "  ELSE  ." + failed " ABORT  THEN ;
```

Words analogous to **+TESTER** could be defined for all the other required words to make up a simple test-suite.

If the data for + is stored in a table a more general form of test word could be written.

```
CREATE +DATA  32767 ,  32767 , 65534 ,
             -32768 , -32768 ,     0 ,
             -32768 ,  65535 , 32767 ,
```

If two variables were created

```
VARIABLE SELECT        and     VARIABLE FUNCTION
```

then the generalized form of the test-word might be

```
: MATH-TEST ( -- )  SELECT @ >R
  R@      @  R@  2 + @  FUNCTION @ EXECUTE  R@  4 + @  =
  R@  6 + @  R@  8 + @  FUNCTION @ EXECUTE  R@ 10 + @  = AND
  R@ 12 + @  R@ 14 + @  FUNCTION @ EXECUTE  R> 16 + @  = AND
  IF ." passed "  ELSE  ." failed " ABORT  THEN ;
```

and the test for + could be made after

```
' +DATA >BODY SELECT !    and   ' + FUNCTION !
```

MATH-TEST could also be used to test - , * , / and other words requiring the same number of arguments and type of test, given the appropriate data tables. This approach would simplify the design of the test-suite considerably and lead to a useful catagorization of words by stack effects and functional type.

Designing a test-suite seems simple, but is it? For one thing, **+TESTER** and **MATH-TEST** rely on a series of other words in order to execute properly, as well as a properly functioning outer interpreter and address interpreter. The word being tested may fail (or pass) for reasons having nothing to do with the word itself. Is it possible to design a test-suite that does not rely to some degree on the rest of the FORTH system? Probably not, because the test-suite could not function without the rest of the FORTH system and its internal code.

But this problem is not as great as it may seem at first. By definition the test-suite would only be used on a FORTH system that is up and running, which itself means that many words function as they should. Even if some tests are invalid because of untested words used in the definition of the test, other tests will probably turn up those words. The inherent redundancy of the test-suite should give it a high reliability. The amount and form of this redundancy will require careful study; writing a good test-suite would be much less trivial than these examples may suggest.

Technical problems are not the only ones that will need to be resolved by the test-suite. Although the definition of **WORD** is explicit in FORTH-83, it is qualified by the sentence

> 'The counted string returned by WORD may reside in the "free" dictionary area at HERE <u>or above</u>.' (emphasis added)

Only careful reading of the standard reveals that the phrase

```
    34 WORD C@ 1+ ALLOT
```

cannot be used to compile a quote-delimited string in a standard program, but instead

```
    34 WORD DUP DUP C@ HERE SWAP 1+ CMOVE C@ 1+ ALLOT
```

must be used to allow for standard dialects that do not parse strings to **HERE** . As most dialects <u>do</u> parse to **HERE** chances seem fairly good that this bug could easily creep into an otherwise standard program. The question is: How many other caveats and exceptions of this sort are there in the standard? The test-suite definitions would provide the definitive answer on how **WORD** and every other required word would have to be used in a standard program. Reducing the potential for this sort of error would be a major benefit of a standard test-suite

Although it will be difficult to write a standard test-suite we feel the effort would be richly repaid. System-authors would have a tool with which to assure compliance with the standard and users would have the means of confirming this compliance. The test-suite would be a library of standard FORTH usage and a guide for students and professionals. And standard FORTH dialects would be more compatible with one another, in keeping with the purpose of the FORTH-83 standard, namely:

> "A standard program shall execute equivalently on all standard systems".