A Proposal For Implementing Local Words In Forth

Ronald L. Greene

Department of Physics
University of New Orleans
New Orleans, LA 70148

Abstract

Some advantages of defining local, headerless words in Forth are presented and discussed briefly. Among these are the use of local words to enhance information hiding and to prevent stack clutter, making code more readable. An approach is presented which uses a local vocabulary to temporarily store the headers of local words until they are linked into the code. The requirements for implementing the approach in a Forth system are discussed.

Introduction

One important use for the parameter stack in Forth is the temporary storage of data. This allows programs to be written without the necessity of defining the many variables that are characteristic of other languages. However, using the stack for temporary storage can be a mixed blessing. Because it is so handy, some Forth programmers tend to completely avoid defining variables for temporary storage, preferring instead to cram data on the stack and to undergo whatever contortion is necessary to work around it. This practice requires a considerable amount of stack manipulation, which can not only make the code difficult to understand, but can also be costly at execution time because of the relative inefficiency of words like `ROT`, `PICK`, etc. This point has been noted by Brodie [BRO84] and by Smith [SMI84], among others.

The judicious use of variables can make Forth code more readable and, under certain circumstances, more efficient. On the other hand, variables have the disadvantage of using additional dictionary space with each declaration. This is especially inefficient if the variable is used only once, or a few times within a single word. Several authors (see, for example, [TOD81]) have suggested defining a class of variables which are stored in the dictionary in a headerless form. In practice variables of this type would be made accessible to one or a few consecutive words, and are consequently called local variables. (Morgenstern [MOR84] has suggested a similar data class, which he refers to as anonymous variables.) Not only would such local variables conserve dictionary space, but they would not be included in dictionary searches, except for the one or few words in which they are used, thus saving compilation and execution time.

In addition to the above advantages, the limited scope of local variables can be helpful in establishing the structure of larger programs through the concept of information hiding [PAR72, BRO84]. Thoughtful application of the concept can free users from the necessity of understanding or recalling the details of the inner workings of a given word. In general, well-defined Forth words should take their input from the stack, perform some operation on it, and put the results back on the stack. If some of the input or output is via global variables, the complexity of the word is increased, which usually leads to more difficulty in program development and maintenance. A change in the value of such a global variable in one part of the code can lead to unexpected effects elsewhere in the

program. A more detailed analysis of global variables has been reported by Wulf and Shaw [WUL73], who compare the potential for misuse of global variables with that of the GOTO statement. Global variables are not necessarily harmful, but their use should be carefully considered. For example, they are best not used for temporary storage of data in order to keep the stack from becoming cluttered. Local variables, however, would be quite helpful for such a task.

The localization concept can be extended to other Forth words as well. In solving a given problem with Forth, intermediate level words are often defined which are referenced by a single, task-specific, high-level word. There is no need for these words to be globally defined. Locally defining them not only would simplify the structure of the program, but, as pointed out above, would save time and memory.

One of the biggest advantages of using local words in Forth would be in its effect on program development by a team of programmers. Because of the global nature of words in Forth, it is currently necessary for different groups of a team to agree on unique names for the words that they will define, or to create separate vocabularies to avoid conflicts. A convenient way of defining local words would eliminate much of the possibility of conflict, since only the highest level words would possess global scope. This form of information hiding would considerably simplify the task of consolidating the work of each group into a single working program.

Although most other modern computer languages routinely use local variables and procedures, at the present time there is no generally accepted method for implementing them in Forth. In the next section I present what I feel is a logical and convenient way of defining and referencing local words in Forth. In practice, the method results in related local words being grouped together in the source code, shortly before the high-level words that refer to them. This helps to reveal their nature to later readers of the program. It also strongly encourages the programmer to consider the scope of words during the design phase, and thus should help guard against unwanted side effects creeping into the program. Previously suggested methods of defining local words allow them to be scattered throughout the source code. This makes it more difficult for readers to comprehend the program structure, and may result in words being made local as an afterthought rather than as an integral part of the design.

The Local Vocabulary

The scheme proposed here allows the creation of not only local variables, but also more general local words. All local words are kept in a single vocabulary called LOCAL. Upon execution of the Forth phrase, LOCAL DEFINITIONS, each newly defined word is added to the LOCAL vocabulary until the next change in the compilation vocabulary. The bodies of local words are stored in the dictionary in the usual way. The headers, however, are stored in a separate memory buffer, and each must end with a pointer to its corresponding body. The local headers, although physically separated from the rest of the dictionary, are linked into the dictionary in such a way that the local vocabulary is searched first. In this way local words will take precedence over global words with the same names.

After one or more local words are defined, non-local words can be created again by changing the compilation vocabulary. When all words which need to access the local words have been defined, the programmer can empty the local header buffer by executing a new LOCAL DEFINITIONS in order to create a new set of local words, or can execute the word END-LOCAL to clear the LOCAL vocabulary without changing the compilation vocabulary. Either action will cause the previously defined local words to be inaccessible. The headers for each new set of words will use the same buffer so that dictionary space is conserved.

For maximum utility, the local vocabulary scheme proposed above should be implemented internally to the Forth system. One way of doing this is described in the next section. The section titled Forth-Level Implementation examines the possibility of implementing the proposal using Forth itself.

Internal Implementation

Three internal variables can be used to implement the local vocabulary approach. One is a flag to indicate whether or not LOCAL is the compilation vocabulary. Execution of the vocabulary word LOCAL should set the local flag to true. Any other vocabulary name (e.g., FORTH) should set it to false. If the local flag is true, the code that creates the header for a new word (usually CREATE) must put the header in the local vocabulary buffer, rather than in the dictionary. The last entry in the local header must be a pointer to the body of the word, which is stored in the dictionary in the usual way.

The second variable used in the scheme should contain a pointer to the next available address within the local buffer. This pointer effectively replaces the dictionary pointer during the creation of a local header, and should be updated in the same way.

To make full use of the local words concept, LOCAL should be searched prior to entering the user-defined vocabulary search order. Thus the LOCAL vocabulary must provide a link to the CONTEXT vocabulary (79 standard) or the transient vocabulary (83 experimental proposal). The start of the local buffer should be referenced by the third internal variable. This would allow the link to the global dictionary to be updated with each new word added to the CONTEXT or transient vocabulary, and in addition, would aid in re-setting the local buffer pointer.

The advantage of using a buffer for the headers of local words is that when there is no longer a need to refer to the words, or when it becomes desirable to "hide" them, the local buffer can be cleared by resetting the buffer pointer. Execution of the words LOCAL or END-LOCAL would reset the pointer, the former in preparation for a new set of local words, and the latter to remove the local words from the search order. Either would again allow access to words of the same name as one or more of those formerly in the LOCAL vocabulary.

Forth Level Implementation

Inclusion of the local vocabulary scheme at the level of Forth requires several additions and changes, some of which depend on the particulars of a given Forth implementation. The description below provides part of what is needed. Hopefully, it will give the reader sufficient information to adapt the local vocabulary approach to his/her own system, and thus to get a feel for how the method works. Let me again emphasize that the local vocabulary scheme is best implemented through changes in the internal operation of Forth. The major reason for this is that the full potential of local words can be reached only if the LOCAL vocabulary is always searched first. Current methods for defining vocabulary search order in Forth do not allow this to be done with the word LOCAL as described above.

To implement local words at the level of Forth requires first the definition of the LOCAL vocabulary. In addition, a constant $LOCBUF (rather than an internal variable), and two variables, $LOCAL and $LP are necessary. The constant $LOCBUF returns the address of the start of the local buffer. The value of $LOCBUF will be system dependent, since the local buffer must not overlap any of the other data structures in the Forth system (such as the parameter and return stacks). The variable $LOCAL contains the flag that indicates whether or not LOCAL is the compilation vocabulary. It will have to be set as necessary by execution of the various vocabulary words. The other variable, $LP, is the local buffer pointer, which points to the next available memory location in the local header. This pointer should be reset to the value of $LOCBUF by the word END-LOCAL.

The Forth-83 definitions for the words described above are as follows:

```
VOCABULARY LOCAL
<address of local buffer> CONSTANT $LOCBUF
VARIABLE $LOCAL
VARIABLE $LP
: FORTH Ø $LOCAL ! FORTH ;
( other vocabularies should be similarly redefined )
: END-LOCAL $LOCBUF $LP ! ;
: LOCAL -1 $LOCAL ! LOCAL ;
```

In order that the headers of `LOCAL` definitions be written in the local buffer rather than the dictionary, the word `CREATE` and words that call it must be redefined. When `$LOCAL` is true, `CREATE` must write the header to the local buffer and add a pointer to the body of the word at the end of the header. Furthermore, words like `]` and `VARIABLE` must be rewritten so that when `$LOCAL` is true they recognize this last cell of the local header as a pointer to the code field address (CFA) rather than the CFA itself. The revised definitions of these words are system dependent, and so are not given here.

Summary

This article has discussed several of the advantages of implementing local words in the Forth language. Chief among these are the use of local words to enhance information hiding, and local variables to prevent stack clutter and to make code more readable by reducing superfluous stack manipulations. Other advantages are the savings in memory and time implicit in headerless words. I have proposed a local vocabulary approach to local words and described briefly how it may be implemented. Although the full power of the approach requires internal changes to the Forth system, its use is naturally consistent with the philosophy of Forth. This work was implemented in an assembly-coded subroutine-threaded version.

References

[BRO84] Leo Brodie, *Thinking Forth*, Prentice-Hall, 1984.

[MOR84] Leonard Morgenstern,"Anonymous Variables", *Forth Dimensions*, Vol. 6, No. 1, FIG, 1984, pp. 33-34.

[PAR72] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", *Communications of the ACM*, Vol. 15, ACM, 1973, pp. 1053-1058.

[SMI84] Rudy Smith, "To Stack or Not. . ." *Forth Dimensions*, Vol 6, No. 2, FIG, 1984, pp. 6-7.

[TOD81] Doug Tody, "Moving FORTH into the Eighties: A Portable Data Reduction System Technical Design Study", *Proceedings of the 1981 Rochester Forth Standards Conference*, May 12-15, 1981, The Institute for Applied Forth Research, Inc., 1981, p. 363.

[WUL73] W. Wulf and Mary Shaw, "Global Variable Considered Harmful", *SIGPLAN Notices*, February, 1973, pp. 28-34.

Manuscript received November, 1984.