 Table of Contents

A.13 The optional Locals word set

The Technical Committee has had a problem with locals. It has been argued forcefully that ANS Forth should say nothing about locals since:

- there is no clear accepted practice in this area;
- not all Forth programmers use them or even know what they are; and
- few implementations use the same syntax, let alone the same broad usage rules and general approaches.

It has also been argued, it would seem equally forcefully, that the lack of any standard approach to locals is precisely the reason for this lack of accepted practice since locals are at best non-trivial to implement in a portable and useful way. It has been further argued that users who have elected to become dependent on locals tend to be locked into a single vendor and have little motivation to join the group that it is hoped will **broadly accept** ANS Forth unless the Standard addresses their problems.

Since the Technical Committee has been unable to reach a strong consensus on either leaving locals out or on adopting any particular vendor's syntax, it has sought some way to deal with an issue that it has been unable to simply dismiss. Realizing that no single mechanism or syntax can simultaneously meet the desires expressed in all the locals proposals that have been received, it has simplified the problem statement to be to define a locals mechanism that:

- is independent of any particular syntax;
- is user extensible;
- enables use of arbitrary identifiers, local in scope to a single definition;
- supports the fundamental cell size data types of Forth; and
- works consistently, especially with respect to re-entrancy and recursion.

This appears to the Technical Committee to be what most of those who actively use locals are trying to achieve with them, and it is at present the consensus of the Technical Committee that if ANS Forth has anything to say on the subject this is an acceptable thing for it to say.

This approach, defining (LOCAL), is proposed as one that can be used with a small amount of user coding to implement some, but not all, of the locals schemes in use. The following coding examples illustrate how it can be used to implement two syntaxes.

The syntax defined by this Standard and used in the systems of Creative Solutions, Inc.:

```
: LOCALS|  ( "name...name |" -- )
    BEGIN
      BL WORD   COUNT OVER C@
      [CHAR] | - OVER 1 - OR
    WHILE
      (LOCAL)
    REPEAT 2DROP   0 0 (LOCAL)
;
IMMEDIATE

: EXAMPLE  ( n -- n n**2 n**3 )
    LOCALS| N |   N  DUP N *  DUP N * ;
```

A proposed syntax: (LOCAL name) with additional usage rules:

```
: LOCAL  ( "name" -- )  BL WORD COUNT (LOCAL) ;  IMMEDIATE

: END-LOCALS  ( -- )  0 0 (LOCAL) ;  IMMEDIATE

: EXAMPLE  ( n -- n n**2 n**3 )
    LOCAL N  END-LOCALS   N  DUP N *  DUP N * ;
```

Other syntaxes can be implemented, although some will admittedly require considerably greater effort or in some cases program conversion. Yet other approaches to locals are completely incompatible due to gross differences in usage rules and in some cases even scope identifiers. For example, the complete local scheme in use at Johns Hopkins had elaborate semantics that cannot be duplicated in terms of this model.

To reinforce the intent of section 13, here are two examples of actual use of locals. The first illustrates correct usage:

a)

```
: {  ( "name ... }" - )
    BEGIN  BL WORD COUNT
      OVER C@ [CHAR] }
      - OVER 1 -  OR
    WHILE
      (LOCAL)
    REPEAT 2DROP 0 0 (LOCAL)
;  IMMEDIATE
```

b)

```
: JOE  ( a b c -- n )
    >R 2* R> 2DUP + 0
    { ANS 2B+C C 2B A }
    2 0 DO  1 ANS + I + TO ANS  ANS . CR  LOOP
    ANS . 2B+C . C . 2B . A . CR
    ANS
;
```

c)

```
100 300 10 JOE .
```

The word { at a) defines a local declaration syntax that surrounds the list of locals with braces. It doesn't do anything fancy, such as reordering locals or providing initial values for some of them, so locals are initialized from the stack in the default order. The definition of JOE at b) illustrates a use of this syntax. Note that work is performed at execution time in that definition before locals are declared. It's OK to use the return stack as long as whatever is placed there is removed before the declarations begin.

Note that before declaring locals, B is doubled, a subexpression (2B+C) is computed, and an initial value (zero) for ANS is provided. After locals have been declared, JOE proceeds to use them. Note that locals may be accessed and updated within do-loops. The effect of interpreting line c) is to display the following values:

1 (ANS the first time through the loop),
3 (ANS the second time),
3 (ANS), 610 (2B+C), 10 (C), 600 (2B), 100 (A),
and 3 (ANS left on the stack by JOE).

The names of the locals vanish after JOE has been compiled. The storage and meaning of locals appear when JOE's locals are declared and vanish as JOE returns to its caller at ; (semicolon).

A second set of examples illustrates various things that break the rules. We assume that the definitions of LOCAL and END-LOCALS above are present, along with { from the preceding example.

d)

```
: ZERO   0 POSTPONE LITERAL POSTPONE LOCAL ; IMMEDIATE
```

e)

```
: MOE  ( a b )
     ZERO TEMP  LOCAL B  1+ LOCAL A+  ZERO ANSWER ;
```

f)

```
: BOB  ( a b c d )  { D C }  { B A } ;
```

Here are two definitions with various violations of rule 13.3.3.2a. In e) the declaration of TEMP is legal and creates a local whose initial value is zero. It's OK because the executable code that ZERO generates precedes the first use of (LOCAL) in the definition. However, the 1+ preceding the declaration of A+ is illegal. Likewise the use of ZERO to define ANSWER is illegal because it generates executable code between uses of (LOCAL). Finally, MOE terminates illegally (no END-LOCALS). BOB in f) violates the rule against declaring two sets of locals.

g)

```
: ANN  ( a b  -- b )  DUP >R  DUP IF { B A } THEN  R> ;
```

h)

```
: JANE  ( a b -- n )  { B A }  A B + >R  A B -  R> / ;
```

ANN in g) violates two rules. The IF ... THEN around the declaration of its locals violates 13.3.3.2b, and the copy of B left on the return stack before declaring locals violates 13.3.3.2c. JANE in h) violates 13.3.3.2d by accessing locals after placing the sum of A and B on the return stack without first removing that sum.

i)

```
: CHRIS  ( a b)
     { B A }  ['] A EXECUTE  5 ['] B >BODY !
     [ ' A ] LITERAL LEE ;
```

CHRIS in i) illustrates three violations of 13.3.3.2e. The attempt to EXECUTE the local called A is inconsistent with some implementations. The store into B via >BODY is likely to cause tragic results with many implementations; moreover, if locals are in registers they can't be addressed as memory no matter what is written.

The third violation, in which an execution token for a definition's local is passed as an argument to the word LEE, would, if allowed, have the unpleasant implication that LEE could EXECUTE the token and obtain a value for A from the particular execution of CHRIS that called LEE this time.

A.13.3 Additional usage requirements

Rule 13.3.3.2d could be relaxed without affecting the integrity of the rest of this structure. 13.3.3.2c could not be.

13.3.3.2b forbids the use of the data stack for local storage because no usage rules have been articulated for programmer users in such a case. Of course, if the data stack is somehow employed in such a way that there are no usage rules, then the locals are invisible to the programmer, are logically not on the stack, and the implementation conforms.

The minimum required number of locals can (and should) be adjusted to minimize the cost of compliance for existing users of locals.

Access to previously declared local variables is prohibited by Section 13.3.3.2d until any data placed onto the return stack by the application has been removed, due to the possible use of the return stack for storage of locals.

Authorization for a Standard Program to manipulate the return stack (e.g., via >R R>) while local variables are active overly constrains implementation possibilities. The consensus of users of locals was that Local facilities represent an effective functional replacement for return stack manipulation, and restriction of standard usage to only one method was reasonable.

Access to Locals within DO..LOOPs is expressly permitted as an additional requirement of conforming systems by Section 13.3.3.2g. Although words, such as (LOCALS), written by a System Implementor, may require inside knowledge of the internal structure of the return stack, such knowledge is not required of a user of compliant Forth systems.

A.13.6 Glossary

A.13.6.1.2295 TO

Typical use: `x TO name`

See: A.6.2.2295 TO

A.13.6.2.1795 LOCALS|

A possible implementation of this word and an example of usage is given in A.13, above. It is intended as an example only; any implementation yielding the described semantics is acceptable.

 Table of Contents

 Next Section