FORTH Dimensions

Volume 7, Number 6

March/April 1986
$4.00

F83 String Functions

The Moving Cursor Writes And, Having Writ, Moves On . . .

Teaching Forth: Let's Keep It Simple

euroFORML '85

FORTH Dimensions
Published by the
Forth Interest Group

Volume VII, Number 6
March/April 1986

Editor
Marlin Ouverson

Production
Cynthia Lawson

Forth Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the Forth Interest Group is in the public domain. Such material may be reproduced with credit given to the author and to the Forth Interest Group.

Subscription to *Forth Dimensions* is free with membership in the Forth Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and to submit material for publication, the address is: Forth Interest Group, P.O. Box 8231, San Jose, California 95155.

Symbol Table

Simple; introductory tutorials and simple applications of Forth.

Intermediate; articles and code for more complex applications, and tutorials on generally difficult topics.

Advanced; requiring study and a thorough understanding of Forth.

Code and examples conform to Forth-83 standard.

Code and examples conform to Forth-79 standard.

Code and examples conform to fig-FORTH.

Deals with new proposals and modifications to standard Forth systems.

FORTH Dimensions

FEATURES

DEPARTMENTS

1986 Rochester Forth Conference

June 11–14, 1986 University of Rochester Rochester, New York

The sixth Rochester Forth Conference is sponsored by the Institute for Applied Forth Research, Inc. in cooperation with the IEEE Computer society and the Laboratory for Laser Energetics of the College of Engineering at the University of Rochester.

The focus of the Conference is on real-time artificial intelligence, systems and applications. The invited speakers will discuss the implementation of a variety of expert systems and their applications, a commercially available data base query system and a real-time version of OPS/5. The performance of high speed Forth engines and moderately parallel execution of rule-based systems will be covered. In addition, presented papers will cover many aspects of implementing and applying Forth and Forth-like languages. These include image processing, instrumentation, robotics, graphics, process control, space-based, medical and business systems. Forth novices, programmers, implementors, and project managers will find these presentations useful and pertinent to their work.

The final day of the Conference will be open to the public, and devoted to tutorials, demonstrations, panel discussions, Forth vendor presentations and poster sessions. All those interested in learning about Forth, or in seeing the most current Forth products available are invited to attend this day at no charge.

The registration fee includes all sessions, meals, and the Conference papers. Lodging is available at local motels or in the UR dormitories. Registration will be from 3-11 pm on Tuesday, June 10th in Wilson Commons, and from 8 am Wednesday, June 11th in Hutchison Hall. There is an hourly shuttle to the airport during registration and checkout. Sessions will be held in Hutchison Hall, and the open day will be in Wilson Commons.

For more information, call or write to Maria Gress at the Institute, 478 Thurston Road, Rochester, New York 14619. Phone: (716)-235-0168.

Registration Form

Name ______________________________

Address ______________________________

____

____

Telephone ______________________________

Registration fees: $325 ________

$275 UR staff, IEEE members

$200 Full time students

Dormitory housing, 5 nights: $100 Single, $75 Double ________

Amount enclosed: $ ________

_____ Vegetarian meal option _____ Non-smoking roommate

Please make checks payable to the Rochester Forth Conference. Mail your registration by May 15th to the Rochester Forth Conference, 478 Thurston Rd., Rochester, NY 14619 USA.

Quirkless CASE?

Dear Marlin:

I really appreciate the cleverness of Michael Jaegermann's letter about the Eaker CASE statement ("A CASE of Pairs," VII/4). After some thought, I came up with a variation on Mr. Jaegermann's idea. His first method required use of a dummy flag to start the **CASE** evaluation, which looked odd. It also did not allow you to mix tests for ranges and single values. His second method required explicit knowledge of the value of **TRUE**, and was not Forth-83 compatible.

The method I present seems to avoid these quirks. My notation may seem strange at first, but I am open to suggestion. The technique is to use **MAX** or **MIN** to generate the proper values to **OF**. For example, if MIN(value,limit) = value, then we know that value < = limit. Screen 13 shows a simple implementation of four tests, each of which will work transparently with **CASE**. Screen 14 shows a sample word to classify ASCII characters. Screen 16 shows how this can be quickly extended to test whether or not a value falls within a range, inclusively or exclusively at either boundary.

Thanks,

Tony Sager
Westminster, Maryland

Sager Screens

```
Screen # 13
( variation on Eaker CASE                        16:46 12/23/85 )
: TASK ;

: ...<=   ( V L -- V L' )   \ if  MIN(V,L)=V then V<=L
   OVER MIN ;

: ...>=   ( V L -- V L' )   \ if MAX(V,L)=l then V>=L
   OVER MAX ;

: ...<   ( V L -- V L' )   \ if MIN(V,L-1)=v then V<L
   1-  ...<=   ;

: ...>   ( V L -- V L' )   \ if MAX(V,L+1)=V then V>L
   1+ ...>=  ;

\ note that MAX MIN give a 'signed' comparison
```

```
Screen # 14
( variation on Eaker CASE                        16:46 12/23/85 )

: CLASSIFY   ( byte -- )
   CASE
    32      ...<  OF ." Control Character"     ENDOF
    ASCII 0 ...<  OF ." Punctuation"           ENDOF
    ASCII 9 ...<= OF ." Digit"                 ENDOF
    ASCII @ ...<= OF ." Punctuation"           ENDOF
    ASCII Z ...<= OF ." Upper Case Letter"     ENDOF
    ASCII a ...<  OF ." Punctuation"           ENDOF
    ASCII z ...<= OF ." Lower Case Letter"     ENDOF
    127     ...<  OF ." Punctuation"           ENDOF
    127           OF ." Rubout Character"      ENDOF
                     ." Not an ASCII Character"
   ENDCASE ;
```

```
Screen # 15
(                                                16:46 12/23/85 )

: TEST   ( hi lo -- )
    DO
       CR
       I 4 .R 3 SPACES I 32 MAX 127 MIN EMIT SPACE
       I CLASSIFY
    LOOP ;
```

Of Extensions and Hotpatches...

Sir:

I would like to add my voice to the opinion expressed by Mark Smiley in *Forth Dimensions* (VII/4) in which he suggests that if Forth is to be adopted by large numbers of users, it will have to be supplied with the facilities that computer users have come to expect. Although I have been programming in Forth for well over three years and have become reasonably facile with the language, when I recently had to write a program that opened a file, read it a line at a time, manipulated these lines (as strings) and eventually wrote them out a byte at a time to a random file, I turned to BASIC. I think there are two separate questions. First, the Forth Standards Team should standardize the use of files, strings, floating-point numbers and other useful tools. Second, vendors should undertake to supply these tools. Only then will Forth have a chance to become the language of choice for all programming tasks.

```
Screen # 16
( RANGE TESTS FOR CASE                              16:46 12/23/85 )

: []   ( V LL UL -- V L' )       \ if MAX(V,LL)=MIN(V,UL) then
   >R OVER MAX R> MIN ;          \    LL <= V <= UL

: ()   ( V LL UL -- V L' )       \ if MAX(V,LL+1)=MIN(V,UL-1) then
   >R OVER 1+ MAX R> 1- MIN ;    \    LL < V < UL

: [)   ( V LL UL -- V L' )       \ if MAX(V,LL)=MIN(V,UL-1) then
   >R OVER MAX R> 1- MIN ;       \    LL <= V < UL

: (]   ( V LL UL -- V L' )       \ if MAX(V,LL+1)=MIN(V,UL) then
   >R OVER 1+ MAX R> MIN ;       \    LL < V <= UL


Screen # 17
( variation on Eaker CASE                           16:46 12/23/85 )

: CLASSIFY2   ( byte -- )
   CASE
    32               ...< OF ." Control Character"     ENDOF
    ASCII 0          ...< OF ." Punctuation"           ENDOF
    ASCII 0 ASCII 9 [] OF ." Digit"                    ENDOF
    ASCII @         ...<= OF ." Punctuation"           ENDOF
    ASCII A ASCII Z [] OF ." Upper Case Letter"        ENDOF
    ASCII a         ...<  OF ." Punctuation"           ENDOF
    ASCII a ASCII z [] OF ." Lower Case Letter"        ENDOF
    127             ...<  OF ." Punctuation"           ENDOF
    127                   OF ." Rubout Character"      ENDOF
                             ." Not an ASCII Character"
   ENDCASE ;


Screen # 18
(                                                   16:46 12/23/85 )

: TEST2   ( hi lo -- )
   DO
     CR
     I 4 .R 3 SPACES I 32 MAX 127 MIN EMIT SPACE
     I CLASSIFY2
   LOOP ;
```

End Sager Screens

Barr Screens

```
Screen # 22
 0 ( subst                                          MB    12/20/85 )
 1 ( Use in the form SUBST WORD1 WORD2 to substitute WORD1 for
 2 WORD2.  The two words may have the same name.)
 3 : SUBST
 4     ?EXEC                                   ( Check if executing)
 5                              ( Get compilation address of word1)
 6     DUP >NAME DUP 32 TOGGLE                        ( Smudge word1)
 7                              ( Get compilation address of word2)
 8     SWAP 32 TOGGLE                               ( Unsmudge word1)
 9     >BODY SWAP OVER !                 ( put cfa of word1 into word2)
10     2+ ['] EXIT SWAP !                 ( put cfa of EXIT into word2)
11     ;
12


Screen # 23
 0 ( Test of subst                                        12/22/85 )
 1 : TESTA CR ." Test A" CR ;
 2 : TESTB CR ." Test B" CR ;
 3 : TEST TESTA ;
 4 TEST ( Prints "Test A")
 5 SUBST TESTB TESTA
 6 TEST ( Prints "Test B")
 7 : TESTA CR ." New test A" CR ;
 8 CR ( Prints "TESTA is redefined")
 9 SUBST TESTA TESTA
10 TEST ( Prints "New test A")
11
12


Screen # 24
 0 ( Mutual recursion test                                12/21/85 )
 1 : A NOOP ;
 2 : B ?DUP IF 1- DUP A 2* SWAP MYSELF +
 3          ELSE 1
 4          THEN ;
 5 : A ?DUP IF 1- DUP A SWAP B +
 6          ELSE 1
 7          THEN ;
 8 SUBST A A
 9
10
11
```

End Barr Screens

I found the idea of redefining words by Phil Koopman, Jr. a very interesting and useful one. When I had difficulty adapting it to my Forth-83 system, I rewrote it to be simpler and more useful (see listing). It is still subject to the limitations mentioned in Mr. Koopman's article: it can be used to redefine colon definitions only, and there must actually be an entry in the word. On the other hand, an unlimited number of words can be redefined, unlike Mr. Koopman's limitation to just one. Also, the syntax is much simpler; you need only type **SUBST WORD1 WORD2** to change all occurrences of **WORD2** to **WORD1**. The two words can even have the same name. In that case, the latest definition is substituted for the penultimate one. Although the word is intended to be used primarily for debugging, there is nothing to stop it from being used, for example, to achieve mutual recursion, as illustrated in the accompanying listing. The words **A** and **B** generate all the pairs of positive integers for which $|B^2 - 2*A^2| = 1$, whose ratios are the best approximations to the square root of two.

The definition uses three words that are not Forth-83 Standard. **?EXEC** merely checks to see if the system is executing, and may be omitted. Lines 6 and 8 use the words **<NAME** and **TOGGLE** that are not standard and that assume words can be smudged to make them invisible to ' (tick). If these three lines are omitted, **SUBST** will work only if **WORD1** and **WORD2** have different names.

Sincerely yours,

Michael Barr
Montreal, Quebec
Canada

Unraveling TI-Forth

Dear Mr. Ouverson:

In *Forth Dimensions* VI/6 (March/April 1985), you published a TI-99/4A screen dump by Howard H. Rogers. I was happy to see some TI-related contributions in *Forth Dimensions*, and noted that you indicated the desire to receive more useful TI utilities to publish.

I gladly offer you this little TI-Forth decompiler program, which is only three screens and not too intimidating for someone to enter by keyboard. Since TI-Forth is an extension of fig-FORTH, I suspect it will work with little or no modification on most such systems.

Some areas that might be implementation dependent are words like **(I")** and **(F-D")**. But those lines can simply be removed for use on another system that doesn't have them.

Using the **DECOMPILER** could hardly be easier. You simply load it and then enter:

DECOMPILE <word name>

and let it rip! It prints the dictionary address of the various component word CFAs and the contents of those words, followed by the symbolic decompilation of the word. I find it very useful to discover just how a lot of the underlying TI-Forth kernel words are implemented.

As an example the readers can easily check out, I have included a printout of the **DECOMPILER**'s output while decompiling one of its own component words.

Sincerely,

Rene LeBlanc
Scottsdale, Arizona

```
 ok
DECOMPILE LSTID

Decompiling: LSTID

Press any key to toggle
  PAUSE/START.

Press <fctn> 4 (BREAK)
  to terminate.


EBE8 A252 CR
EBEA A53E DUP
EBEC B376 U.
EBEE A53E DUP
EBF0 A574 @
EBF2 B376 U.
EBF4 A53E DUP
EBF6 A574 @
EBF8 A5BE 2+
EBFA AA4C NFA
EBFC AF24 ID.
EBFE A446 ;S
 ok

DECOMPILE DECOMPILE

Decompiling: DECOMPILE

Press any key to toggle
  PAUSE/START.

Press <fctn> 4 (BREAK)
  to terminate.


EE28 AE9E -FIND
EE2A A060 0BRANCH A
EE2E A51C DROP
EE30 EC0E (DECOMPILE)
EE32 A050 BRANCH 1E
EE36 A252 CR
EE38 ADA6 (.")
    Word not in dictionary"
EE52 A252 CR
EE54 A446 :S
 ok
```

LeBlanc Screens

```
SCR #154
 ( TI-FORTH DECOMPILE )   ( DECOMPILE word )
  0 CLOAD DECOMPILE BASE->R  DECIMAL
 : LSTID CR DUP U. DUP @ U. DUP @ 2+ NFA ID. ;
 : (DECOMPILE) ( pfa -- ) [COMPILE] BASE->R HEX CR
   DUP NFA CR ." Decompiling: " ID. CR CR
   ." Press any key to toggle PAUSE/START." CR
   ." Press <fctn> 4 (BREAK) to terminate."  CR CR
   BEGIN ?KEY UNTIL
   DUP 2- @ [ ' CONSTANT 6 + ] LITERAL =
   IF DUP U. DUP @ U.  ." CONSTANT " NFA ID. ELSE
      DUP 2- @ [ ' VARIABLE 6 + ] LITERAL =
      IF DUP U. DUP @ U.  ." VARIABLE " NFA ID. ELSE
         DUP 2- @ [ ' USER 6 + ] LITERAL =
         IF DUP U. DUP @ U.  ." USER " NFA ID. ELSE
            DUP 2- @ [ ' VOCABULARY 30 + ] LITERAL =
            IF DUP U. ."      VOCABULARY " NFA ID. ELSE [ -->


SCR #155
 ( TI-FORTH DECOMPILE - cont. )

  ] BEGIN DUP      @ 2+ [ ' ;S       ] LITERAL = PAUSE OR NOT
    WHILE DUP      @ 2+ [ ' LIT      ] LITERAL =
          OVER     @ 2+ [ ' 0BRANCH ] LITERAL = OR
          OVER     @ 2+ [ ' BRANCH   ] LITERAL = OR
          OVER     @ 2+ [ ' (LOOP)   ] LITERAL = OR
          OVER     @ 2+ [ ' (+LOOP) ] LITERAL = OR
          OVER     @ 2+ [ ' (OF)     ] LITERAL = OR  [ -->


SCR #156
 ( TI-FORTH DECOMPILE - cont. )
  ]        IF   LSTID 2+ DUP @ .
           ELSE DUP  @ 2+ [ ' (.")   ] LITERAL =
                OVER @ 2+ [ ' (F-D") ] LITERAL = OR
                OVER @ 2+ [ ' (!")   ] LITERAL = OR
                IF   LSTID 2+ DUP
                     COUNT SWAP OVER TYPE ASCII " EMIT
                     SPACE + 1- =CELLS
                ELSE LSTID THEN  THEN 2+
    REPEAT LSTID DROP
   THEN THEN THEN THEN [COMPILE] R->BASE ;

   : DECOMPILE -FIND IF DROP (DECOMPILE) ELSE
     CR ." Word not in dictionary" THEN CR ;

  R->BASE
```

End LeBlanc Screens

Editorial

Subjective Benchmark

The last issue in our membership year is only a subjective sort of benchmark, at least insofar as it affects the publishing schedule. (The next issue will, after all, show up in another two months as usual.) But it is a good time to work in a little self-analysis between reviewing manuscripts, copyediting, keyboarding, uploading files to the typesetter and managing the process of producing *Forth Dimensions*. And it's a great time to thank our authors and all FIG members for their unflagging support and contribution, be it in the form of articles, criticism or just appreciation.

Some truly fine articles are already on file for upcoming issues, and we look forward to reviewing many new manuscripts from our readers. We plan to continue the fine tutorials by Michael Ham, John James and others. Users of TI-Forth will be seeing some material specific to their systems, thanks to recent contributions. Of course, the emphasis will continue to be on code in Forth-83 and Forth-79, with some fig-FORTH material as well.

We look forward to hearing from many of you in the coming months. *Forth Dimensions* is very much "by and for" FIG members, and you can keep it that way with your active participation. Keep those letters and articles coming!

—Marlin Ouverson
Editor

Index to Advertisers

The Moving Cursor Writes And, Having Writ, Moves On . . .

Michael Ham
Santa Cruz, California

But if you press the up-arrow,
It goes back and rewrites the line.
(Apologies to both Omar Khayyam and Edward FitzGerald.)

Items are usually selected from a program menu in one of two ways: by the user entering some identifying information (e.g., the first character of the option selected, or the option number), or by the user moving the cursor to the desired item and then pressing carriage return or "enter" (or that odd little symbol that made so much sense to IBM, displayed on the screen by the sequence **17 EMIT 196 EMIT 217 EMIT**).

The word that accommodates the selection normally returns an option number, which is used by the program to execute the proper function. This article describes one approach to selecting from a menu: picking the option with the cursor and using the resulting option number as an index into an array of functions.

Because the elements of this task are specific to the machine and the Forth used, you may have to translate some of the terms to match your particular resources. For example, the IBM does not deliver cursor key information the same way the Apple II does; and the command for cursor placement is not the same in PC/Forth (by Laboratory Microsystems, Inc.) as it is in polyFORTH (by FORTH, Inc.). The code shown in this article was written in PC/Forth for an IBM PC or compatible.

Block 1

Block 1 defines the true/false constants and redefines two of PC/Forth's words. **PCKEY** works like **KEY** for all the regular ASCII keys, leaving their ASCII value on the stack. For the special keys (the function and cursor keys), **PCKEY** leaves a false flag on top of the stack and the IBM key value of the special key beneath. For example, **PCKEY** leaves 75 0 on the stack if left-arrow is pushed.

Because I generally prefer that a word return always the same number of arguments on the stack, I defined **MYKEY** to leave a true flag above the normal ASCII keys. The word **0<>** converts any non-zero value to a true flag, leaving a zero (false flag) unchanged.

The cursor-placement word in PC/Forth is **GOTOXY**, which expects the column number (the x-coordinate) followed by the row number (the y-coordinate) — that is, the row number on top of the stack. This approach doubtless satisfies half the users, but the other half will agree with me that row number first, then column number, is clearly the natural order. I thus define my own cursor placement word, mimicking the polyFORTH word in action as well as name. The ability to "fix up" native commands to meet one's own needs (prejudices?) is one of the most attractive features of Forth.

Your Forth probably has some way to turn the (actual) cursor off. Normally you don't want the cursor blinking away wherever it last landed, while the user contemplates the menu. Some Forths automatically extinguish the cursor while **KEY** waits for a key; others provide an explicit cursor attribute word. PC/Forth's **SET-CURSOR** allows you to define the height of the cursor. **-CURSOR** uses **SET-CURSOR** to define the height away altogether, so that the cursor vanishes. **+CURSOR** restores the cursor on exit.

BELL is my idea of how the "error" bell should sound. You can tune it to your own taste by changing the parameters given to the PC/Forth word **BEEP**.

Block 2

The cursor location can be shown by any of several tactics: a "pointer" character (the IBM has various character symbols useful for this purpose), underlining or inversing the current option (returning it to normal mode when the cursor moves on), changing the color of the current option, and so forth.

Since any of the options can be selected and thus can differ from the others, you must be able to write each option by itself. Block 2 contains a collection of words to write each option. An additional space is included before and after the text in each option because I used inverse video to show the selected option, and the extra space makes the inverse look better, particularly if you are using the IBM color graphics adapter.

The header **OPTIONS** is put into the dictionary with **CREATE**, and then **]** is used to turn the compiler on. We use the compiler to put the compilation address of each of the following words into the dictionary. **[** turns the compiler off again. The effect is the same as if the **]** and **[** had not been used and instead we had ticked and comma'd each word into the dictionary:

```
CREATE OPTIONS  ' "1 ,  ' "2 ,  ' "3 ,
```

But **]** and **[** take less room, look better and are easier to read. **OPTIONS** names an array of compilation addresses: the addresses of the words that write the various options to the screen.

Block 3

Block 3 contains words to manipulate the options and the option numbers. Given an option number, **COL1?** and **COL2?** tell whether the number is in column one or column two by comparing the number of the option against half the number of options. This example has six options, numbered zero through five; three (one-half of six) is the option number of the first option in the second half — that is, the first option in the second column.

CLIP uses **MOD** to coerce any number into the range of legal option numbers — **6 CLIP** produces a zero; **-1 CLIP** produces a five. As we add to or subtact from the option number on the stack, we can **CLIP** the result to make it the appropriate option number within the legal range (for this example) of 0 through 5.

PLAIN, given an option number, dips into the array **OPTIONS** and executes the word that displays that option. The option number is multiplied by two because each address in the **OPTIONS** array is two bytes long. Because the sequence **@ EXECUTE** is so common, many Forths provide some specific word for it. PC/Forth has **PERFORM** and polyFORTH has **@EXECUTE** (spelled without the space).

INVERSE distinguishes the choice of the moment in inverse video through the use of the PC/Forth word **REVERSE**. You can redefine **INVERSE** to distinguish the chosen option in whatever way you prefer: color, underline, capitalization vs. lower-case, etc.

SHOWALL displays all the options, with option 0 shown as the current choice.

COLSWAP, given the option number of the currently selected option, first redisplays that option in the **PLAIN** format (in effect unselecting it) and then converts the number to the number of the option in the same position in the other column. This option is then displayed by **INVERSE** as the current choice.

Because there are only two columns in this example, we can move from one column to another simply by adding the column length (which is one-half the number of options for the two-column case) to the option number. If there were more than two columns, we would have to decide whether to subtract the column length (to move left) or add the columnn length (to move right). After adding, **CLIP** insures that the sum is a legal option number.

The ForthCard™

STAND ALONE OPERATION

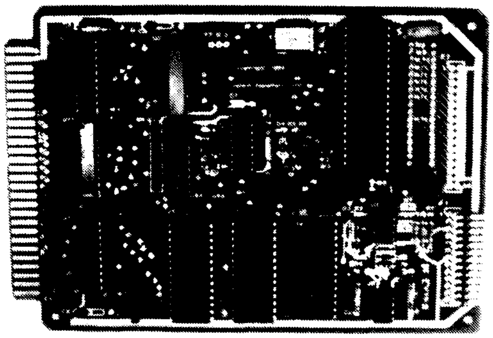
STD BUS INTERFACE

EPROM/EEPROM
PROGRAMMER

RS-232 I/O

PARALLEL I/O

ROCKWELL FORTH CHIP



Evaluation Unit **$299**
Part #STD65F11-05 includes:
ForthCard, Development
ROM, 8Kbyte RAM, Manuals

OEM Version as low as
Part #STD65F11-00 **$199**
does not include
memory or manuals

The Forthcard provides OEMs and end users with the ability to develop Forth and assembly language programs on a single **STD bus compatible** card.

Just add a CRT terminal (or a computer with RS-232 port), connect 5 volts and you have a **self contained Forth computer.** The STD bus interface makes it easy to expand.

Download Forth source code using the serial port on your PC. Use the **onboard EPROM/EEPROM programming** capability to save debugged Forth and assembly language programs. Standard UV erasable EPROMs may also be programmed with an external Vpp supply.

NEW! Options and Application Notes

Electrically Eraseable PROMs (EEPROMs)

FREEZE the dictionary in EEPROM (save in non-volatile memory, to be restored on power up)

Download Software for your IBM PC or CP/M

Non-Volatile CMOS RAM with battery 2K, 8K, optional Clock/calendar

Fast 2MHz clock (4MHz crystal)

Disk Controller Card (5¼")

Self Test Diagnostics

Parallel printer interface

Ask about our ForthBox™
A complete STD bus oriented system including the ForthCard, Disk Controller, Disk Drive(s), STD Card Cage, Cabinet and power supply.

CALL TODAY FOR COMPLETE INFORMATION!

HiTech Equipment Corporation

9560 Black Mountain Road
San Diego, CA 92126
(619) 566-1892

VISA

Block 4

This block contains constants that correspond to the values returned by the keys of interest: up, down, left, right, and carriage return. **MYKEY** leaves two numbers on the stack (the key value and the flag); I treat them as one double-precision number.

Block 5

GETOPTION is a long word with a simple structure. I could have used PC/Forth's **CASE** statement to make it easier to read, but the function would be the same. The word **CLS** clears the screen.

The program comments note that this version allows the cursor to wraparound in the up-down movement, but not in the left-right direction. This inconsistency was to illustrate both options; it would be removed in any actual application.

You can allow left-right wraparound by making the changes noted in the comments. As an exercise, try revising the word to eliminate wraparound in the up-down direction. This can be a little tricky, depending on how you decide you want it to work. You might want the up key, for example, to produce a bell but no movement if the cursor is on option number 0 or option number 3. Obviously, you should implement the restriction by reference to the contents of the variable **#OPTIONS** rather than to a numeric literal.

Block 6

Block 6 creates another set of words. **OPT**n is a placeholder for the task accomplished by option number n. The last option, the "quit" choice, probably would be **BYE** (which leaves Forth and returns to the operating system) in the final program, but during development **ABORT** is more convenient; you also need to relight the cursor. (In some IBM clones, the cursor automatically reappears when the program exits to DOS; in others, not.)

In **TASKS** we see another array of words; **RUN** (the final program word) uses the option number (left on the stack by **GETOPTION**) to pick from **TASKS** the appropriate word to execute. Note the similarity of the function done by **TASK** and **OPTIONS**: both are used as headers at definition time and later, at run time, both find themselves involved in the same sort of computation: a computation involving **2* + @** and **EXECUTE**. This suggests a defining word; the **CREATE** part is simple, and the **DOES>** part should also be easy. Remember that **DOES>** puts the defined word's address on the stack at run time, so that you will need a **SWAP** before the **2***. As is often the case, the tricky part is finding a good name for the defining word.

```
Block 1
( Basic tools                                    Ham 10:27 02/08/86 )

    0 CONSTANT F            -1 CONSTANT T

: MYKEY (  - c f ) ( f = T if ASCII character, F if special )
     PCKEY ?DUP O<> ;

: TAB  ( col# row# - )  SWAP GOTOXY ; ( positions cursor )

: -CURSOR  14  0 SET-CURSOR ;   ( removes cursor )

: +CURSOR   6  7 SET-CURSOR ;   ( returns cursor )

: BELL  512 ( freq ) 8 ( duration )  BEEP ;   ( can be tuned )

Block 2
( Sample option selection                        Ham 10:28 02/08/86 )

: "1   5 15 TAB ."  First option " ;
: "2   7 15 TAB ."  Second option " ;
: "3   9 15 TAB ."  Third option " ;
: "4   5 50 TAB ."  Fourth option " ;
: "5   7 50 TAB ."  Fifth option " ;
: "6   9 50 TAB ."  QUIT option " ;   ( always offer a way out )

 CREATE OPTIONS ]  "1  "2  "3  "4  "5  "6  [

 VARIABLE #OPTIONS ( holds number of options )

Block 3
( Option arithemtic                              Ham 10:34 02/08/86 )

: COL1?  ( # - f ) ( T = optn in left col ) #OPTIONS @ 2/ < ;

: COL2?  ( # - f ) ( T = optn in right col ) COL1? NOT ;

: CLIP ( # - #' ) #OPTIONS @ MOD ; ( keeps no. in legal range)

: PLAIN ( # - ) 2* OPTIONS + @ EXECUTE ;  ( prints option )

: INVERSE ( # - ) REVERSE PLAIN REVERSE ; ( option inversed )

: SHOWALL   0 INVERSE  #OPTIONS @ 1  DO I PLAIN LOOP ;

: COLSWAP ( # -#') DUP PLAIN #OPTIONS @ 2/ + CLIP DUP INVERSE ;

Block 4
( Key identification                             Ham 10:27 02/08/86 )

( The 75, 77, 72, and 80 are the key numbers returned by the
  IBM.  The false flag on top in effect makes the number double
  precision.  -65523. is shorthand for 13 under a true flag. )

: L?  ( d - f ) 75. D= ;       ( T if left arrow pressed )
: R?  ( d - f ) 77. D= ;       ( T if right arrow pressed )
: UP? ( d - f ) 72. D= ;       ( T if up arrow pressed )
: DN? ( d - f ) 80. D= ;       ( T if down arrow pressed )
: CR? ( d - f ) -65523. D= ;   ( T if carriage return pressed )
```

Because **RUN** ends with **F UNTIL**, completing a task returns to the main menu. (Most Forths, including PC/Forth, provide the word **AGAIN** as a synonym for **F UNTIL**.) The program repeats until the user selects the "quit" word, which breaks out of the loop. (Another approach is to end the loop by fetching a value from a variable — e.g., **SWITCH @ UNTIL** — and have the "quit" task's sole job being to store a "true" (-1) into the variable **SWITCH**.)

The approach illustrated in this example can be used for a wide variety of menu-based programs. The separate individual tasks can, of course, present their own menus, with subtasks associated with each of those menu options. As an exercise, revise **GETOPTION** so that it can be used by these subsidiary menus as well as by the main menu. Some of the revisions you will want to consider are controlling the number put into **#OPTIONS** (so that each subsidiary menu can initialize it to the appropriate value before calling **GETOPTION**) and altering **PLAIN** (so that it does not assume a particular array but instead takes the array address from a variable).

Michael Ham is a freelance programmer, systems designer and writer in Santa Cruz, California. This article is from a book in progress. Copyright© 1986 by Michael Ham.

```
Block 5
( Sample option selection                        Ham 10:34 02/08/86 )
: GETOPTION ( - # ) 6 #OPTIONS ! CLS -CURSOR SHOWALL 0  BEGIN
   MYKEY 2DUP UP? IF 2DROP DUP PLAIN 1- CLIP DUP INVERSE F
   ELSE  2DUP DN? IF 2DROP DUP PLAIN 1+ CLIP DUP INVERSE F
   ELSE  2DUP L?  IF 2DROP DUP COL1? IF BELL ELSE COLSWAP THEN F
   ELSE  2DUP R?  IF 2DROP DUP COL2? IF BELL ELSE COLSWAP THEN F
   ELSE  2DUP CR? IF DROP ( what's left will act as true flag )
   ELSE  2DROP BELL F THEN THEN THEN THEN THEN UNTIL ;
 ( leaves option # on stack; first option is option # 0 )
 ( This version sounds a bell if user attempts to move left
   and is in left column.  It could instead wrap the cursor
   by replacing COL1? IF BELL ELSE COLSWAP THEN with COLSWAP )

Block 6
( Options jobs                                   Ham 10:14 02/08/86 )
: OPTSTATEMENT   ( n - ) CLS -CURSOR 10 36 TAB ." Option " .
   12 34 TAB ." Press a key."  BELL KEY DROP ;
: OPT1  1 OPTSTATEMENT ;  ( fake jobs for illustration )
: OPT2  2 OPTSTATEMENT ;
: OPT3  3 OPTSTATEMENT ;
: OPT4  4 OPTSTATEMENT ;
: OPT5  5 OPTSTATEMENT ;
: OPT6  +CURSOR BYE ( use +CURSOR ABORT during development ) ;
  CREATE TASKS ]  OPT1 OPT2 OPT3 OPT4 OPT5 OPT6 [
: RUN  BEGIN GETOPTION 2* TASKS + @ EXECUTE F UNTIL ;
```

euroFORML '85

Robert Reiling, President
Forth Interest Group

—*Stettenfels Castle* A castle in West Germany was the location of the International euroFORML Conference held October 25 through 27, 1985. Attending the conference were sixty-six people from ten countries. All the sessions were conducted in English in order to provide a common language for this diversified group. Of course, Forth was the common computer language. The conference agenda included conference papers, workshops, poster sessions, a panel discussion, hardware and software demonstrations, and time for independent discussions.

Stettenfels Castle is located near Heilbronn in southern Germany, overlooking agricultural land that is predominantly vineyards. Gardens about the castle are attractive and offered the opportunity for a pleasant walk during breaks in the conference program. The castle has sleeping accommodations, and many conference attendees stayed there throughout the event. All meals were prepared by the castle staff and were served in the dining hall. Late-night discussion groups met in front of the fireplace in the castle tower.

Klaus Schleisiek and other dedicated Forth enthusiasts in Germany organized the euroFORML Conference. They sent promotional material throughout Europe, which resulted in over fifty attendees from Europe registering for the conference. Thirteen registered from the United States. Klaus appointed Michael Perry, from the United States, to moderate the conference.

Conference papers were interesting and informative. A brief look at these papers follows.

English as a Second Language for Forth Programmers
Wil Baden

There is a difference between spelling a word and saying a word. With this statement as an opening remark, Wil Baden presented his ideas about "saying" such Forth words as **#**, @, **!**, **+!**, **[**, **C!** and others. Baden suggests that C! should be pronounced "byte set"; @ would be "value." These are only examples of Baden's proposal. This paper caused a great deal of discussion among the attendees.

Interpretive Logic
Wil Baden

This paper presents a method for conditional execution, conditional compilation and text editing, which extends Forth to be responsive to modern system requirements. Forth source screens are included to demonstrate the principles he proposed.

Data Collection in Elementary Particle Physics with 32-bit VAX/68K Forth
R. Haglund

Forth is used to control large-scale data collection systems. This paper discusses the application and then explains why Forth is suitable for applications in physics. Haglund points out that one can optimize both development and execution speed to the most suitable level.

In-Situ-Development: The Ideal Complement to Cross-Target-Compiling
A.P. Haley
H.P. Oakford
C.L. Stephens

This paper describes a package called "In-Situ-Development," which aims to provide an easily implemented technique to allow developers of stand-alone applications hardware and software to take advantage of cross-target-compilers without losing direct contact with their hardware.

Forth and Artificial Intelligence
Robert LaQuey

This is a progress report on work with Forth to implement the minimum set of concepts needed for the support of artificial intelligence. Several screens of Forth code demonstrate the progress in this effort.

A Forth-Driven Network System for Applied Automation
D.C. Long

The availability of low-cost, single-board computers and intelligent input/output systems provides exciting new capabilities for the automation of systems and entire facilities. This paper describes an applications command language, implemented in Forth, known as the "Master Control Program." Supported hardware presently includes the Optomux family of intelligent interface boards.

Performance Analysis in Threaded-Code Systems
M.A. Perry

Perry states that a good rule of thumb is that a program spends ninety percent of its time executing ten percent of the code. When some performance goal must be met, it is necessary to find those routines in which most time is spent and make them run faster. Forth encourages modular programming, and it is easy to replace a slow routine once it has been found. Several techniques for performance analysis in Forth systems are described in this paper.

Generic Operators
T. Rayburn

The paper presents techniques for writing Forth programs that have resulted in dramatic improvement to the readability of code. Presented are current implementations and some ideas for future work.

Control Simulation for a Tape Deck
L. Richter-Abraham

The code for control of a stereo tape deck is developed in this simulation example. A "virtual tape deck" is used to check the code. The ideas presented in this paper could be used to develop a training program for control simulation.

Preliminary Report on the Novix 4000
C.L. Stephens
W.P. Watson

The Novix 4000 is a true Forth processor and is capable of ten million Forth instructions per second. It is implemented as a gate array. This paper introduces the architecture of the chip, its hardware configuration and the software support provided with it. Application areas are suggested for the chip.

A Set of Forth Words for Electrical Network Analysis
J. Storjohann

The program presented in this paper uses a simple approach to describe components and networks, and to immediately invoke suitable arithmetic operations. This approach avoids the numbering of nodes and the storing and inverting of large matrices. The whole system, including the complex floating-point words, occupies about two kilobytes.

Forth Language Extension for Controlling Interactive Jobs on Other Machines
D.K. Walker

A Forth application on an IBM PC/XT is described for 1) collecting, editing and generating input for a large model of the Norwegian economy used by the Norwegian government; 2) transferring this information to a mainframe; and 3) running interactive jobs which check the input, process it further and send it on to another mainframe where the economic model equations are solved and result tables are written. The application emulates a person operating a computer terminal. Forth techniques illustrated include a finite-state, description-language extension for controlling general, interactive jobs on other machines.

RTDF: A Real-Time Forth System Including Multi-tasking
H.E.R. Wijnands
P.M. Bruijn

This paper outlines a real-time Forth system intended for use as a development tool for single-processor control systems. Due to the general language concepts applied, it has also proved useful for discrete system simulation and other concurrent programming needs. It offers multiple task declaration, initiation and priority assignment.

Event-Driven Multi-tasking: A Syntax
J. Zander

In this paper situations are investigated where, for various reasons, interrupts cannot be used. An example is when the condition tested is a very complex one. A Forth syntax for general event handling is proposed, including the structures **EVERY**, **AFTER** and **WHENEVER ... PERFORM**. An implementation for (time-shared) multi-tasking Forth is sketched.

All of the euroFORML papers described above and the complete 1985 FORML papers from the USA conference at Asilomar, are included in the 1985 FORML Proceedings. *This book is available from the Forth Interest Group.*

FORTH INTEREST GROUP MAIL ORDER FORM

P.O. Box 8231 San Jose, CA 95155 (408) 277-0668

MEMBERSHIP

IN THE FORTH INTEREST GROUP

107 - MEMBERSHIP in the FORTH INTEREST GROUP & Volume 8 of FORTH DIMENSIONS. No sales tax, handling fee or discount on membership. See the back page of this order form.

The Forth Interest Group is a worldwide non-profit member-supported organization with over 4,000 members and 90 chapters. FIG membership includes a subscription to the bi-monthly publication, FORTH Dimensions. FIG also offers its members publication discounts, group health and life insurance, an on-line data base, a large selection of Forth literature, and many other services. Cost is $30.00 per year for USA, Canada & Mexico; all other countries may select surface ($37.00) or air ($43.00) delivery.

The annual membership dues are based on the membership year, which runs from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of Forth Dimensions and subsequent issues will be mailed to you as they are published.

You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

HOW TO USE THIS FORM

1. Each item you wish to order lists three different Price categories:

   Column 1 - USA, Canada, Mexico
   Column 2 - Foreign Surface Mail
   Column 3 - Foreign Air Mail

2. Select the item and note your price in the space provided.
3. After completing your selections enter your order on the fourth page of this form.
4. Detach the form and return it with your payment to **The Forth Interest Group**.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May - April)

101 - Volume 1 FORTH Dimensions (1979/80) $15/16/18 ______
102 - Volume 2 FORTH Dimensions (1980/81) $15/16/18 ______
103 - Volume 3 FORTH Dimensions (1981/82) $15/16/18 ______
104 - Volume 4 FORTH Dimensions (1982/83) $15/16/18 ______
105 - Volume 5 FORTH Dimensions (1983/84) $15/16/18 ______
106 - Volume 6 FORTH Dimensions (1984/85) $15/16/18 ______
107 - Volume 7 FORTH Dimensions (1985/86) $15/16/18 ______

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Assembly Language Source Listings of fig-Forth for specific CPUs and machines with compiler security and variable length names.

513 - 1802/MARCH 81 .... $15/16/18 ______
514 - 6502/SEPT 80 .... $15/16/18 ______
515 - 6800/MAY 79 .... $15/16/18 ______
516 - 6809/JUNE 80 .... $15/16/18 ______
517 - 8080/SEPT 79 .... $15/16/18 ______
518 - 8086/88/MARCH 81 .... $15/16/18 ______
519 - 9900/MARCH 81 .... $15/16/18 ______
520 - ALPHA MICRO/SEPT 80 .... $15/16/18 ______
521 - APPLE II/AUG 81 .... $15/16/18 ______
522 - ECLIPSE/OCT 82 .... $15/16/18 ______
523 - IBM-PC/MARCH 84 .... $15/16/18 ______
524 - NOVA/MAY 81 .... $15/16/18 ______
525 - PACE/MAY 79 .... $15/16/18 ______
526 - PDP-11/JAN 80 .... $15/16/18 ______
527 - VAX/OCT 82 .... $15/16/18 ______
528 - Z80/SEPT 82 .... $15/16/18 ______

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH .... $25/26/35 ______
Glen B. Haydon
An annotated glossary for MVP Forth; a 79-Standard Forth.

205 - BEGINNING FORTH .... $17/18/21 ______
Paul Chirlian
Introductory text for 79-Standard.

215 - COMPLETE FORTH .... $16/17/20 ______
Alan Winfield
A comprehensive introduction including problems with answers. (Forth 79)

220 - FORTH ENCYCLOPEDIA .... $25/26/35 ______
Mitch Derick & Linda Baker
A detailed look at each Fig-Forth instruction.

225 - FORTH FUNDAMENTALS, V. 1 .... $16/17/20 ______
Kevin McCabe
A textbook approach to 79-Standard Forth.

230 - FORTH FUNDAMENTALS, V. 2 .... $13/14/16 ______
Kevin McCabe
A glossary.

232 - FORTH NOTEBOOK .... $25/26/35 ______
Dr. C. H. Ting
Good examples and applications. Great learning aid. PolyFORTH is the dialect used. Some conversion advice is included. Code is well documented.

233 - FORTH TOOLS .... $20/21/24______
Gary Feierbach & Paul Thomas
The standard tools required to create and debug Forth-based applications.

235 - INSIDE F83 .... $25/26/35 ______
Dr. C. H. Ting
Invaluable for those using F-83.

237 - LEARNING FORTH .... $17/18/21 ______
Margaret A. Armstrong
Interactive text, introduction to the basic concepts of Forth. Includes section on how to teach children Forth.

240 - MASTERING FORTH .... $18/19/22 ______
Anita Anderson & Martin Tracy
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.

245 - STARTING FORTH (soft cover) .... $20/21/24 ______
Leo Brodie
A lively and highly readable introduction with exercises.

246 - STARTING FORTH (hard cover) .... $24/25/29 ______
Leo Brodie

255 - THINKING FORTH (soft cover) .... $16/17/20 ______
Leo Brodie
The sequel to "Starting Forth". An intermediate text on style and form.

265 - THREADED INTERPRETIVE LANGUAGES $23/25/28 ______
R.G. Loeliger
Step-by-step development of a non-standard Z-80 Forth.

270 - UNDERSTANDING FORTH .... $3.50/5/6 ______
Joseph Reymann
A brief introduction to Forth and overview of its structure.

FORML CONFERENCE PROCEEDINGS

FORML PROCEEDINGS - FORML (the Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group

310 - **FORML PROCEEDINGS 1980 .... $30/33/40 ______**
Technical papers on the Forth language and extensions.

311 - **FORML PROCEEDINGS 1981 (2V) ...... $45/48/50 ______**
Nucleus layer, interactive layer, extensible layer, metacompilation, system development, file systems, other languages, other operating systems, applications and abstracts without papers.

312 - **FORML PROCEEDINGS 1982 .... $30/33/40 ______**
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - **FORML PROCEEDINGS 1983 .... $30/33/40 ______**
Forth in hardware, Forth implementations, future strategy, programming techniques, arithmetic & floating point, file systems, coding conventions, functional programming, applications.

314 - FORML PROCEEDINGS 1984 .... $30/33/40 ______
Expert systems in Forth, using Forth, philosophy, implementing Forth systems, new directions for Forth, interfacing Forth to operating systems, Forth systems techniques, adding local variables to Forth.

ROCHESTER PROCEEDINGS

The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.

321 - ROCHESTER 1981 (Standards Conference) $25/28/35 ______
79-Standard, implementing Forth, data structures, vocabularies, applications and working group reports.

322 - ROCHESTER 1982
(Data bases & Process Control) .... $25/28/35 ______
Machine independence, project management, data structures, mathematics and working group reports.

323 - ROCHESTER 1983 (Forth Applications) . $25/28/35 ______
Forth in robotics, graphics, high-speed data acquisition, real-time problems, file management, Forth-like languages, new techniques for implementing Forth and working group reports.

324 - ROCHESTER 1984 (Forth Applications) . $25/28/35 ______
Forth in image analysis, operating systems, Forth chips, functional programming, real-time applications, cross-compilation, multi-tasking, new techniques and working group reports.

325 - ROCHESTER 1985
(Software Management and Engineering) $20/21/24 ______
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L, and a Forth-based business applications language, includes working group reports.

THE JOURNAL OF FORTH APPLICATION & RESEARCH

A refereed technical journal published by the Institute for Applied Forth Research, Inc.

401 - JOURNAL OF FORTH RESEARCH V.1 #1 $15/16/18 ______
Robotics.

402 - JOURNAL OF FORTH RESEARCH V.1 #2 $15/16/18 ______
Data Structures.

403 - JOURNAL OF FORTH RESEARCH V.2 #1 $15/16/18 ______
Forth Machines.

404 - JOURNAL OF FORTH RESEARCH V.2 #2 $15/16/18 ______
Real-Time Systems.

405 - JOURNAL OF FORTH RESEARCH V.2 #3 $15/16/18 ______
Enhancing Forth.

406 - JOURNAL OF FORTH RESEARCH V.2 #4 $15/16/18 ______
Extended Addressing.

407 - JOURNAL OF FORTH RESEARCH V.3 #1 $15/16/18 ______
Forth-based laboratory systems and data structures.

REPRINTS

420 - BYTE REPRINTS .... $5/6/7 ______
Eleven Forth articles and letters to the editor that have appeared in *Byte* magazine.

421 - POPULAR COMPUTING 9/83 .... $5/6/7 ______
Special issue on various computer languages, with an in-depth article on Forth's history and evolution.

DR. DOBB'S JOURNAL

This magazine produces an annual special Forth issue which includes source-code listings for various Forth applications.

422 - DR. DOBB'S 9/82....................... $5/6/7 ______

423 - DR. DOBB'S 9/83....................... $5/6/7 ______

424 - DR. DOBB'S 9/84....................... $5/6/7 ______

425 - DR. DOBB'S 10/85 .... $5/6/7 ______

HISTORICAL DOCUMENTS

501 - KITT PEAK PRIMER.................. $25/27/35 ______
One of the first institutional books on Forth. Of historical interest.

502 - FIG-FORTH INSTALLATION MANUAL .. $15/16/18 ______
Glossary model editor - We recommend you purchase this manual when purchasing the source-code listings.

503 - USING FORTH .... $20/21/23 ______
FORTH, Inc.

REFERENCE

305 - FORTH 83 STANDARD .... $15/16/18 ______
The authoritative description of 83-Standard Forth. For reference, not instruction.

300 - FORTH 79 STANDARD .... $15/16/18 ______
The authoritative description of 79-Standard Forth. Of historical interest.

316 - BIBLIOGRAPHY OF FORTH REFERENCES
2nd edition, Sept. 1984 .... $15/16/18 ______
An excellent source of references to articles about Forth throughout microcomputer literature. Over 1300 references.

MISCELLANEOUS

601 - T-SHIRT SIZE ______________
Small, Medium, Large and Extra-Large.
White design on a dark blue shirt. $10/11/12 ______

602 - POSTER (BYTE Cover) .... $15/16/18 ______

616 - HANDY REFERENCE CARD .... FREE ______

683 - FORTH-83 HANDY REFERENCE CARD ...... FREE ______

FORTH MODEL LIBRARY

The model applications disks described below are new additions to the Forth Interest Group's library. These disks are the first releases of new professionally developed Forth applications disks. Prepared on 5 1/4"disks , they are IBM MS DOS 2.0 and up compatible. The disks are compatible with Forth-83 systems currently available from several Forth vendors. Macintosh 3 1/2"disks are available for MasterFORTH systems only.

Forth-83 Compatibility IBM MS DOS

Laxen/Perry F83 — LMI PC/FORTH 3.0
MasterFORTH 1.0 — TaskFORTH 1.0
PolyFORTH®II

Forth-83 Compatibility Macintosh

MasterFORTH

ORDERING INFORMATION

701 - Volume 1 - **A Forth List Handler**
by Martin J. Tracy $40/43/45______
Forth is extended with list primitives to provide a flexible high-speed environment for artificial intelligence. ELISA and Winston & Horn's micro-LISP are included as examples. Documentation is included on the disk.

702 - Volume 2 - **A Forth Spreadsheet**
by Craig A. Lindley $40/43/45______
This model spreadsheet first appeared in Forth Dimensions Volume 7, Issue 1 and 2. These issues contain the documentation for this disk.

703 - Volume 3 - **Automatic Structure Charts** by Kim R. Harris
$40/43/45______
These tools for the analysis of large Forth programs were first presented at the 1985 FORML conference. Program documentation is contained in the 1985 FORML Proceedings.

Please specify disk size when ordering.

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 408/277-0668

Name ______________________
Title ______________________
Company ______________________
Address ______________________
City ______________________
State/Prov. ____________ **Zip** ____________
Country ______________________
Phone ______________________

OFFICE USE ONLY

By ________ Date ________ Type ________
Shipped By ________ Date ________
UPS Wt. ________ Amt. ________
USPS Wt. ________ Amt. ________
BO Date ________ Wt. ________ Amt. ________
By ________

ITEM #	TITLE	AUTHOR	QTY	UNIT PRICE	TOTAL
107	MEMBERSHIP →				SEE BELOW

SUBTOTAL	
10% MEMBER DISCOUNT MEMBER #	
SUBTOTAL	
CA. RESIDENTS SALES TAX	
HANDLING FEE	**$2.00**
MEMBERSHIP FEE ☐ NEW ☐ RENEWAL $30/37/43	
TOTAL	

☐ Check enclosed (payable to: **FORTH INTEREST GROUP**)

☐ VISA ☐ MASTERCARD

Card # ______________________

Expiration Date ______________________

Signature ______________________

($15.00 minimum on charge orders)

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
P.O. Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: Monday-Friday, 9am-5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US Bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within five days of receipt of the order. Please allow 4-6 weeks for out-of-stock books (delivery in most cases will be much sooner).

SALES TAX
Deliveries to Alameda, Contra Costa, San Mateo, Los Angeles, Santa Cruz Counties, add 6½%: Santa Clara County, add 7%; other California Counties, add 6%.

4/86

Teaching Forth:
Let's Keep It Simple

Ronald E. Apra
San Jose, California

For as long as I have taught beginning programming to elementary and secondary students, the IF THEN construct has been a source of confusion and frustration for beginning students. The problem is further compounded when you introduce the ELSE statement along with IF THEN. Students who have taken geometry before taking programming seem to handle this construct much better due to their experience with IF THEN statements in formal proofs. However, if geometry is made a prerequisite for programming, you limit the number of students who can take programming.

I feel that the source of this confusion lies partly in the syntax of the IF THEN ELSE phrase (BASIC, Logo and Pascal) or the more mind-boggling IF ELSE THEN expression used in Forth. Apple Logo and IBM Logo offer two interesting ways to deal with the problem. In the first method, the operation IF is followed by a test that produces a true or false flag followed by one or two instruction lists. If the flag is true the first instruction is executed, and if the flag is false the second instruction is run. This method eliminates the words THEN and ELSE. In the following Logo procedure,

```
TO DECIDE
OUTPUT IF 0 = RANDOM 2 ["YES]
  ["NO]
END
```

the 0 = RANDOM 2 after the IF produces a "TRUE flag if RANDOM 2 is 0 and the procedure will OUTPUT the message YES. If 0 = RANDOM 2 is "FALSE, then the procedure will run the second list, which is equivalent to the ELSE phase of a conditional. In the second method, the Logo commands TEST, IFTRUE and IFFALSE can be used to write the procedure DECIDE in the following way:

```
TO DECIDE
TEST 0 = RANDOM 2
IFTRUE [OUTPUT "YES]
IFFALSE [OUTPUT "NO]
END
```

where a very readable procedure is produced. TEST yields a "TRUE or "FALSE flag for IFTRUE and IFFALSE.

Since some of my students do program in Forth, it would be a good exercise for them to see if they could come up with a construct in Forth that would work similarly to the above Logo procedure. I am not at this time proposing that the standard IF ELSE THEN construct in Forth be changed, but I challenge the Forth community to see what they can come up with.

In playing around with this problem in Forth, I have written the four simple words **TESTIT**, **IFTRUE**, **IFFALSE** and **ENDIT**. **TESTIT** is defined simply as

```
: TESTIT  ( n -- n n ) DUP ;
```

and can be used to duplicate a flag or some number on the stack that is about to be tested. In **EXAMPLE1** on screen 95, **TESTIT** is duplicating the flag produced by **0=** and in **EXAMPLE4** on screen 96, **TESTIT** is duplicating the number on the stack that is about to be tested by **1 =**. **IFTRUE** is a new name for **IF**, and **ENDIT** is a new name for **THEN** (see *Forth Dimensions* VI/1, page 26, for a different version of **ENDIT**). I defined **IFTRUE** and **IFFALSE** as follows:

```
: IFTRUE ( flag -- ) [COMPILE] IF ; IMMEDIATE
: IFFALSE ( flag -- ) [COMPILE] THEN ; IMMEDIATE
```

On screen 95, **EXAMPLE1** resolves an IF ELSE THEN condition with a **TESTIT IFTRUE IFFALSE** structure. By the nature of the syntax, the student can point out the **0=** test of n and knows, if the flag is true, where the true condition will be executed. For beginning students, the IF ELSE THEN syntax does not leave enough clues to where the parts of the conditional should go. When a student gets some practice with the **TESTIT IFTRUE IFFALSE** construct, he can better understand the **IFTRUE ELSE ENDIT** or **IFFALSE ELSE ENDIT** structure in **EXAMPLE2** and **EXAMPLE3** of screen 95.

On screen 96, **EXAMPLE4** can produce some interesting results where **TESTIT** is duplicating the input to be tested by **1 =**. See if you can explain why **3 EXAMPLE4** outputs "twothreefour" and then create your own crazy **EXAMPLE**. I bet there are some interesting things that can be done with **TESTIT** and multiple **IFTRUE** and **IFFALSE** statements. In **EXAMPLES5**, the words **TESTIT**, **IFTRUE** and **ENDIT** seem to improve the readability of the nesting, but I try to encourage students to avoid nesting if at all possible.

I stress "keep it simple" in my programming philosophy, but most beginning students are overwhelmed by the articles that appear in *Forth Dimensions*. For example, the "Techniques Tutorial" department seemed to be a showcase for the skills of some truly great programmers, but it was over the heads of many beginners. I hope this article will stimulate more thought along the lines of "keeping it simple."

```
SCR #95
 ( TESTIT, IFTRUE, IFFALSE, ENDIT                     ra/Oct/1985 )

 : TESTIT ( n --  n n ) DUP ;
 : ENDIT [COMPILE] THEN ; IMMEDIATE
 : IFTRUE ( flag -- ) [COMPILE] IF ; IMMEDIATE
 : IFFALSE ( flag -- ) COMPILE NOT [COMPILE] IF ; IMMEDIATE

 : EXAMPLE1 ( n -- )
     0= TESTIT
           IFTRUE ." true" ENDIT
           IFFALSE ." false" ENDIT ;

 : EXAMPLE2 ( n -- ) 0= IFTRUE ." true" ELSE ." false" ENDIT ;

 : EXAMPLE3 ( n -- ) 0= IFFALSE ." false" ELSE ." true" ENDIT ;
SCR #96
 ( TESTIT, IFTRUE, IFALSE, ENDIT        ra/0ct/85 )

 : EXAMPLE4 ( n -- )
      TESTIT 1 = IFTRUE ." one" ENDIT
      TESTIT 2 = IFFALSE ." two" ENDIT
      TESTIT 3 = IFTRUE ." three" ENDIT
      TESTIT 4 = IFFALSE ." four" ENDIT
              DROP ;

 : EXAMPLES5 ( n -- )
      TESTIT 1 = IFTRUE ." one" ELSE
      TESTIT 2 = IFTRUE ." two" ELSE
      TESTIT 3 = IFTRUE ." three" ELSE
      TESTIT 4 = IFTRUE ." four" ELSE
        ENDIT ENDIT ENDIT ENDIT DROP ;
OK
1 EXAMPLE4 onetwofour
2 EXAMPLE4 four
3 EXAMPLE4 twothreefour
4 EXAMPLE4 two
```

F83 String Functions

Clifford Kent
Mottville, New York

I was drawn to the Laxen and Perry F83 public-domain Forth model by its meta-compiler, full source code and many innovations. The following is a language extension in support of that model, for use when programming time is more important than program execution speed.

The string functions that follow were developed from those presented in *BYTE* by John Cassady ("Stacking Strings in Forth," 1980, reprints available from the Forth Interest Group). Those interested in a more complete description of how a string stack works should see that article. While some words are taken directly from that article, others have been changed to use the facilities of F83, and many new functions have been added. This string function package brings to Forth the ease of text handling usually found in more limited (or is it limiting?) languages like Pascal or BASIC.

Three forms of string storage are used. String constants can be compiled into the Forth dictionary for use at run time. String variables compile named buffers into the dictionary. A string stack is located in high memory outside the Forth dictionary, between the dictionary top and the parameter stack, for use in manipulating strings. As strings are added, it grows downward toward the dictionary.

The action of the string stack is parallel to the action of the parameter stack. I have made a conscious effort to keep these string functions consistent with their numeric equivalents. All string functions are directed at the top of the string stack and/or the top of the parameter stack. A string constant places its characters on the string stack just as a numeric constant places its value on the parameter stack. The run-time action of a string variable is to place an address on the parameter stack. When you fetch a string variable, its characters are copied to the string stack top. When the top string is saved in a string variable, it is also removed from the string stack. String constant arrays and string variable arrays expect a zero-based index on the parameter stack. The string variable array returns an address. A string constant array moves the string constant to the string stack.

In order to store variable-length strings on the string stack, each string of characters on the string stack is preceded by a sixteen-bit string length. This limits maximum string length to 65,535 characters. Some of the functions in the package assume signed integers in their error testing; these functions will only operate correctly with strings of less than 32,765 characters. I have not found this to be a problem. Since there are no checks of the contents of the strings handled by these words, they may also be used to manipulate data records of any data type. For example, if a temporary array is needed, just use **$CHRS** to create a string of the correct length, filled with any character. Use **$P@ 2+** to find the address of the first byte in the array. **$DROP** reclaims the space when done.

Many words specific to the Laxen and Perry F83 Forth Model have been used here. While performance is quite good, these words are not extremely portable. The smart **MOVE** is used where needed to avoid problems with overlap. **LENGTH** is the sixteen-bit equivalent of **COUNT**. **SCAN** searches for the first occurrence of a character. **UPPER** converts lower case to upper case. **TUCK** can be replaced by **DUP ROT SWAP**. **BETWEEN** does a ranged test. **NUMBER?** converts a string to a double number and a success/failure flag. **COMPARE** does a < = > test of two strings.

A glossary, full source code with shadow screens and an index to the source are included here.

The CP/M version of F83 positions the block buffers, return stack and parameter stack in memory each time it is loaded from disk. (See KERNEL80.BLK screen 85 for the F83 cold-start code.) In order for pre-compiled systems using a string stack to be portable, the string stack must be positioned relative to the parameter stack each time it loads. **$P-INIT** does this initialization. It should be included as part of the system or application initialization. (See EXTEND80.BLK screen 2 for the word **HELLO**.) I normally allow 512 bytes for the parameter stack; to allow more or less space, change the definition of **$P-INIT** in screen 16.

Top$** and **Sec$ are very handy for adding new string functions. They include error checking, and return an address and length suitable for use by **$@**, **TYPE**, **CMOVE**, **CMOVE>** or **MOVE**.

The sub-string functions **$POS**, **$DELETE** and **$COPY** use one (not zero) to point to the first character in a string. This, along with the testing done by **$DELETE** and **$COPY**, allows the results of a **$POS** search to be used directly, without **IF THEN** statements, to trap errors. For example,

```
ASCII  ,  $POS
1  SWAP  $COPY
```

will search the top string for a comma and copy all characters up to and including the comma to a new top string. If no comma is found, a null string will be created. Or, you could use:

```
ASCII  ,  $POS
?DUP  IF
1 SWAP  $COPY  THEN
```

to avoid the creation of the null string.

$IN needs a maximum string length on the parameter stack. It uses **EXPECT** to get a string from the terminal and pushes it onto the string stack. The F83 version of **EXPECT** is unusually flexible: it uses an execution array to decode control characters. The variable **CC** holds a pointer to this array, so the editing functions available can be changed by creating a new execution array and changing **CC**. In this way, the action of **$IN** can be redefined as required for different functions.

String variables store the buffer size when compiled. **$VAR!** uses this number when saving a string. The actual string length is saved for use by **$VAR@**. Thus, strings are only stored to the length of the variable's buffer, and are fetched in their original length if the string was shorter than the variable's buffer. Note that the string variable buffer is cleared to blanks in preparation for each string save. This allows alternate versions of **$VAR@** to fetch fixed-length strings for output in fixed-length fields. It also allows an entire database record to be assembled in a string variable's buffer.

Nearly all of the standard Forth number-printing words have been translated to create strings instead. Their use should be clear. **$DOLLARS** is a useful, special-case word. It converts a double number (assumed to be dollars x 100) into a right-justified string of specified length with a leading dollar sign. It calls the more general number formatter **(decimal$D.R)** that can be used to create other specialized number formats.

The word **$=** uses the F83 word **COMPARE** to test the top two strings up to the length of the shorter string. **CAPS** is tested before each string compare. If **CAPS** is true, both strings are converted to upper case before comparing the strings.

There is only a little error checking in these words, but it is generally adequate to prevent total system destruction. For those who want no error trapping:

STRINGS Vocabulary

Fifty-one names have been defined:

```
$!             in: TOOLS.BLK    screen: 17
$"             in: TOOLS.BLK    screen: 21
$+             in: TOOLS.BLK    screen: 18
$,             in: TOOLS.BLK    screen: 18
$=             in: TOOLS.BLK    screen: 30
$@             in: TOOLS.BLK    screen: 17
$CHRS          in: TOOLS.BLK    screen: 19
$CONST-ARRAY   in: TOOLS.BLK    screen: 24
$CONSTANT      in: TOOLS.BLK    screen: 24
$COPY          in: TOOLS.BLK    screen: 20
$D.            in: TOOLS.BLK    screen: 25
$D.L           in: TOOLS.BLK    screen: 25
$D.R           in: TOOLS.BLK    screen: 25
$DELETE        in: TOOLS.BLK    screen: 20
$DOLLARS       in: TOOLS.BLK    screen: 26
$DROP          in: TOOLS.BLK    screen: 17
$DUP           in: TOOLS.BLK    screen: 18
$IN            in: TOOLS.BLK    screen: 21
$LC->UC        in: TOOLS.BLK    screen: 27
$N.            in: TOOLS.BLK    screen: 25
$N.L           in: TOOLS.BLK    screen: 25
$N.R           in: TOOLS.BLK    screen: 25
$OVER          in: TOOLS.BLK    screen: 18
$P             in: TOOLS.BLK    screen: 16
$P!            in: TOOLS.BLK    screen: 16
$P-INIT        in: TOOLS.BLK    screen: 16
$P0            in: TOOLS.BLK    screen: 16
$P0$P!         in: TOOLS.BLK    screen: 16
$P2@           in: TOOLS.BLK    screen: 16
$P@            in: TOOLS.BLK    screen: 16
$POS           in: TOOLS.BLK    screen: 19
$STRIP         in: TOOLS.BLK    screen: 28
$SWAP          in: TOOLS.BLK    screen: 19
$TRIM          in: TOOLS.BLK    screen: 28
$U.            in: TOOLS.BLK    screen: 25
$UC->LC        in: TOOLS.BLK    screen: 27
$VAL           in: TOOLS.BLK    screen: 29
$VAR!          in: TOOLS.BLK    screen: 23
$VAR-ARRAY     in: TOOLS.BLK    screen: 22
$VAR@          in: TOOLS.BLK    screen: 23
$VARFILL       in: TOOLS.BLK    screen: 23
$VARIABLE      in: TOOLS.BLK    screen: 22
$str           in: TOOLS.BLK    screen: 25
$var_build     in: TOOLS.BLK    screen: 22
(char-test)    in: TOOLS.BLK    screen: 20
(decimal$D.R)  in: TOOLS.BLK    screen: 26
?1$P@          in: TOOLS.BLK    screen: 16
?2$P@          in: TOOLS.BLK    screen: 16
Sec$           in: TOOLS.BLK    screen: 17
Top$           in: TOOLS.BLK    screen: 17
["]            in: TOOLS.BLK    screen: 21
```

- replace **?1$P@** and **?2$P@** with **$P@** and **$P2@**.
- remove **(char-test)** from **$DELETE** and **$COPY**.
- be very careful when running new code.

I have also included my string stack debugging tools in screen 35. For a clear understanding of the string stack in operation, I suggest using each word in a simple example, then dumping the string stack or string variable to see what has happened in memory.

The functions presented here are by no means a complete set, and additions will be welcomed. These words are compiled in 1813 bytes by F83. As presented here, the code is optimized for size and ease of use, not for maximum execution speed. I have tried to maintain functional grouping within the source so that parts of the package could be used when dictionary space is tight.

Finally, I would like to express my thanks to Hank Fay and the members of the Central New York FIG Chapter for their encouragement and constructive criticism.

Glossary

$PO (S -- addr)

A constant that points to the string stack base. I normally allow for 512 bytes of stack RAM. Some applications will need more, others less.

$P (S -- addr)

A variable that holds the address of the current string stack top.

$P-INIT (S --)

String stack initialization routine. Make this word part of your system or application startup. This is needed because F83 positions the block buffers, return stack and parameter stack in memory each time it loads. **$P-INIT** positions the string stack by checking the stack pointer base. To change the size of the parameter stack, change the 512 in this word to the stack size needed.

POP! (S --)

Clear the string stack by resetting the string stack pointer.

$P! (S addr --)

Save a new string stack pointer.

$P@ (S -- addr)

Fetch the string stack pointer. Returns the address of the length of the top string.

$P2@ (S -- addr)

Fetch a pointer to the second string. Returns the address of the length of the second string.

?1$P@ (S -- addr)

Fetch the string stack pointer. Aborts with an error message if the string stack is empty.

?2$P@ (S -- addr)

Fetch the pointer to the second string. Aborts with an error message if the string stack does not contain two strings.

$DROP (S --)
Drop top string.

$@ (S addr len --)
Fetch to the string stack the string whose address and length are on the parameter stack.

Top$ (S -- addr len)
Returns the address of the first character and the length of the top string.

Sec$ (S -- addr len)
Returns the address of the first character and the length of the second string.

$! (S addr --)
Pop the top string to the address on the parameter stack. The string length is not moved with the string. This is not for use with string variables.

$. (S --)
TYPE the top string to the current output device. Like a number on the parameter stack, the top string is lost.

$DUP (S --)
Duplicate the top string.

$+ (S --)
Combine the top two strings into one string. The second string will be added to the end of the top string.

$OVER (S --)
Copy second string and push it on the string stack.

$SWAP (S --)
Swap the top two strings.

$POS (S c -- pos | 0)
Search the top string for the character on the parameter stack. If not found, return a zero; if found, return the position of the character. The first character is number one (not zero). The output of **$POS** may be used directly by **$DELETE** and **$COPY**.

$CHRS (S len c --)
Makes a new string of specified length, filled with character c. (This need not be a printable character.)

(char-test) (S pos cnt -- pos' cnt')
Error trap routine used by **$DELETE** and **$COPY**. This will prevent most big errors by changing pos and cnt to legal values for the current top string.

$DELETE (S pos cnt --)
Delete cnt characters from the top string, starting at pos. The input string is destroyed. Impossible input will result in no change to the string. The string's characters are numbered starting at one (not zero).

$COPY (S pos cnt --)
Make a new string at the top of the string stack by copying part of the old top string. The copied string starts at pos and includes cnt characters. The old top of string stack is not changed. Impossible input creates a null string. The string's characters are numbered starting at one (not zero).

$IN (S n --)
A simple input line editor that gets a string of maximum length n from the terminal and leaves it on the string stack. This uses the F83 version of **EXPECT**, which can be redefined to change the functions of the input editor by changing the execution array pointed to by the variable **CC** or by changing **CC** to point to a different execution array.

$VARIABLE (S compile: $len --)
(S run-time: -- addr)

Used in the form

15 $VARIABLE <name>

to create a new string variable with space for fifteen characters. When <name> is executed, it returns the address of the length (sixteen bits) of the currently stored string. The maximum length of a string variable's buffer is stored at addr–2.

$VAR-ARRAY (S compile: $len size --)
(S run-time: n -- addr)

Used in the form

15 8 $VARIABLE <name>

to create a new array of string variables <name> with space for eight strings of fifteen characters each. When <name> is executed, it converts element number n to the address of the length (sixteen bits) of the currently stored string. The maximum length of each of the string array variable's buffers is stored at addr–2.

$VARFILL (S addr c --)
Fill the string variable at addr with character c.

$VAR@ (S addr --)
Fetch the string at addr and push it on the string stack. Since the string's actual length is stored with the characters, only the characters originally saved will be returned. If the string variable is empty, a null string will be returned.

$VAR! (S addr --)
Pop the top string from the string stack and save it in the string variable at addr. The string variable's buffer is first cleared to blanks. The actual string length is saved with the characters for later use. If the top string is too long for the string variable's buffer, it will be truncated on the right.

["] (S --)
Used by **$"** to move an in-line compiled string to the string stack.

$" (S --)
If compiling, compile the string that follows in-line to be moved to the string stack at execution time. If executing, put the enclosed string on the string stack. Used in the form:

(**$"** File not found")

$CONSTANT (S compile: --)
(S run-time: --)
A defining word that compiles named string constants. At compile time, create an initialized string constant. At run time, move the constant to the string stack top. Example:

$CONSTANT TITLE "Annual Report"

where **TITLE** would place 'Annual Report' on the string stack. Note: Use one blank followed by a double quote after the name of the **$CONSTANT**. **WORD** is used to compile the string up to the second double quote, and **WORD** is very picky about leading blanks and delimiters. However, this allows blanks to be compiled into the array. Because **WORD** returns an eight-bit length, the maximum length of a string constant is 256 characters.

$CONST-ARRAY (S compile: $len --)
(S run-time: n --)
A defining word that compiles a named array of string constants. At compile time, create an initialized array of string constants. At run time, move element n to the string stack top. Example:

6 $CONST-ARRAY
NAME "Cliff Janet LaurenKent "

where **1 NAME** would put 'Janet' on the string stack. Note: Use one blank followed by a double quote after the array name. **WORD** is used to compile the string up to the second double quote, and **WORD** is very picky about leading blanks and delimiters. However, this allows blanks to be compiled into the array. Because **WORD** returns an eight-bit length, the maximum length of a string constant array is 256 characters.

The following words parallel the standard Forth number formatting words. Each creates a string on the string stack. If the field width specified will not contain the number, the string will be longer than specified; no data is lost.

Stack for following: (S d field --)
$D.L 32-bit left justified
$D.R 32-bit right justified

Stack for following: (S d --)
$D. 32-bit signed

Stack for following: (S n field --)
$N.L 16-bit left justified
$N.R 16-bit right justified

Stack for following: (S n --)
$U. 16-bit unsigned
$N. 16-bit signed

(decimal$D.R) (S d-num field places --)

Convert a double number to a right-justified string with 'field' characters and 'places' digits after the decimal.

$DOLLARS (S d-num field --)
Using field width at TOS, convert the double number/100 to a string as dollars and cents. Note that the dollar sign and decimal point are included in the character count, so there are two digits less than the field width. If the field width will not contain the number, the string will be longer than specified; no data will be lost.

$LC->UC (S --)
Replace all lower case with upper case.

$UC->LC (S --)
Replace all upper case with lower case.

$TRIM (S --)
Remove trailing blanks from top string. This is the string stack equivalent of **-TRAILING**.

$STRIP (S --)
Remove leading blanks from top string.

$VAL (S -- d f)
Converts the top string to a double number, using the current system base. The string is lost. A leading minus sign is allowed. Leading and trailing blanks are also allowed; however, no blanks are allowed between a minus sign and the number that follows. The system variable **DPL** will contain the number of characters to the right of the decimal, if any. The flag at TOS indicates the success or failure of the conversion.

$= (S -- f)

Compare the two top strings to the length of the shorter string. The flag may take any of three values:

0 – the strings are equal
1 – the top string is shorter
– 1 the top string is longer
Neither string is lost or altered.

The following words have been handy while writing string handling routines. They are normally excluded from the run-time system.

CLR$S (S --)
Clears the top 256 bytes of the string stack to zeroes, making debugging with **.$S** easier.

.$S (S --)
A non-destructive dump of the top 256 bytes of the string stack area in hex format. This will show string contents, string order and the string lengths.

.$V (S addr --)
Displays a string variable in memory.

```
    15
\ Load String Stack                                05/10/85ck

 CAPS OFF
 DECIMAL
 5 VIEW# !
CR .( String Stack = )  HERE
 ONLY FORTH ALSO DEFINITIONS      VOCABULARY STRINGS
 STRINGS ALSO DEFINITIONS
  1 15 +THRU             ( Basic string stack words )
HERE SWAP - U.
\ 35 LOAD              CR .( String dump words loaded.)
CAPS ON
\S
```

```
    16
\ basic stack words                                05/08/85ck
0 CONSTANT $P0
  VARIABLE $P
: $P-INIT                        (S  --   )
          SP0 @ 512 -  ['] $P0 >BODY !  $P0 $P ! ;
: $P0$P!   $P0 $P ! ;            (S  --   )
: $P!      $P ! ;                (S addr --   )
: $P@      $P @ ;                (S   -- addr )
: $P2@                           (S   -- addr )
           $P @ LENGTH + ;
: ?1$P@                          (S   -- addr )
           $P @ DUP $P0 U>=
           IF  $P0$P! ." String Stack Empty." ABORT  THEN ;
: ?2$P@                          (S   -- addr )
           $P2@ DUP $P0 U>=
           IF  $P0$P! ." Need Two Strings." ABORT  THEN ;
```

```
    17
\ basic stack words                                03/25/85CK

: $DROP                          (S  --   )
        ?1$P@ DUP @ + 2+ $P ! ;
: $@                             (S addr len --   )
        DUP >R  $P @ SWAP -  SWAP OVER R@ MOVE
        2 - R> OVER ! $P ! ;
: Top$                           (S   -- addr len )
        ?1$P@ LENGTH ;
: Sec$                           (S   -- addr len )
        ?2$P@ LENGTH ;
: $!                             (S addr --   )
        Top$ ROT SWAP  MOVE  $DROP ;
```

```
    66
                                                   03/26/85ck
 The basis for this is an article in BYTE by John Cassady.
Many words are taken from that article.  The action of these
string words is parallel to the action of FORTH's parameter
stack words. Strings are brought to the $stack for use.
Funtions are directed at the top of the $stack.  There is very
little error checking in these words, but for those who want
none, replace ?1$P@ and ?2$P@ with $P@ and $P2@.  Since there
are no checks on the contents of the 'strings' handled by these
words, they may also be used to manipulate data records of any
data type.  The smart CMOVE is used here to avoid problems with
overlap.  In addition, many words specific to the Laxen & Perry
F83 FORTH MODEL have been used. While performance has been
enhanced, this version is not as portable as the fig-FORTH
version.
```

```
    67
                                                   05/08/85ck
$P0 - constant returning the address of the $stack base.
$P - variable holding the address of the $stack top
$P-INIT - positions the $stack 512 bytes below parameter stack
$P0$P!                        (S  --   )
  Clear the $stack by reseting the $stack pointer.
$P!                           (S addr --   )
  Save a new $stack top address.
$P@                           (S  -- addr )
  Fetch the $stack top address.
$P2@                          (S  -- addr )
  Fetch the address of the 2nd string.
?1$P@                         (S  -- addr )
  Fetch the $stack top address - error if no string.
?2$P@                         (S  -- addr )
  Fetch the address of the 2nd string - error if not 2 strings.
```

```
    68
                                                   03/25/85CK
$DROP                         (S  --   )
  Drop top string.
$@                            (S addr len --   )
  Fetch the string whose address and length are on the P-stack
  to the $stack.
Top$                          (S  -- addr len )
  Returns the address and length of the top string.
Sec$                          (S  -- addr len )
  Returns the address and length of the second string.
$!                            (S addr --   )
  Pop the top string to the address on the P-stack. The string
  length is not moved with the string. This is not for use
  with string variables.
```

```
   18
\ basic stack words                               03/26/85ck

: $.                        (S   --  )
      Top$ TYPE $DROP ;

: $DUP                      (S   --  )
      Top$ $@ ;

: $+                        (S   --  )
      Sec$ Top$  ROT OVER + >R
      OVER 2+ SWAP CMOVE>
      DROP  $P@ 2+ $P!  R> $P@ ! ;

: $OVER                     (S   --  )
      Sec$ $@ ;
```

```
   69
                                                  03/25/85CK
$.                          (S   --  )
  Output the top string to the current device. Like a number
  on the P-stack, the top string is lost.

$DUP                        (S   --  )
  Duplicate the top string.

$+                          (S   --  )
  Combine the two top strings into one string. The second
  string will be added to the end of the top string.

$OVER                       (S   --  )
  Copy second string and push it on the $stack.
```

```
   19
\ $SWAP $POS $CHRS                                06/15/85ck

: $SWAP                     (S   --  )
      $OVER  Top$ DUP 2+ >R   Sec$ SWAP DROP  + 4 +
      SWAP 2- DUP R@ + ROT CMOVE>  $P@ R> + $P! ;

: $POS                      (S c -- pos | 0 )
      Top$ DUP >R  ROT SCAN ?DUP
      IF    R> SWAP - 1+  SWAP DROP
      ELSE  R> 2DROP FALSE
      THEN ;

: $CHRS                     (S len c --   )
      SWAP 0 MAX  DUP $P@  SWAP - 2- $P!
      $P@ !  $P@ LENGTH ROT FILL ;
```

```
   70
                                                  05/07/85ck
$SWAP                       (S   --  )
  Swap the top two strings. First the second string is copied
  to the top, then the two top strings are moved in memory to
  pack the stack, then the $pointer is corrected for the move.

$POS                        (S c -- pos | 0 )
  Search the top string for the character on the P-stack. If
  not found return a 0, if found return the position of the
  character. The first character is number 1 (not 0). The
  output of $POS may be used directly by $DELETE and $COPY.

$CHRS                       (S len c --   )
  Makes a new string of specified length, filled  with
  character c.
```

```
   20
\ $DELETE $COPY                                   05/13/85ck

: (char-test)               (S pos cnt -- pos' cnt' )
      2 ?ENOUGH   OVER ?1$P@ @ >
      IF  DROP 0  THEN
      OVER $P@ @ SWAP - 1+ MIN   0 MAX ;

: $DELETE                   (S pos cnt --   )
      (char-test)  DUP >R  SWAP 1+ >R
      $P@ DUP ROT + R> CMOVE>
      R> DUP $P +!  $P@ @ SWAP - $P@ ! ;

: $COPY                     (S pos cnt --   )
      (char-test)
      SWAP $P@ 1+ +  SWAP $@ ;
```

```
   71
                                                  05/07/85ck
(char-test)                 (S pos cnt -- pos' cnt' )
  Error trap routine used by $DELETE and $COPY. This will
  prevent most big errors by changing pos and cnt to legal
  values.

$DELETE                     (S pos cnt --   )
  Delete cnt characters from the top string, starting at pos.
  The input string is destroyed.

$COPY                       (S pos cnt --   )
 Make a new string at the top of the $stack by copying part of
 the old top. The copied string starts at pos and includes cnt
 characters. The old top of $stack is not changed. Impossible
 input creates a null string.
```

```
    21
\ $" $IN                                             05/07/85ck

: ["]                          (S  --  )
       R@ DUP 2+ SWAP @  DUP 2+ R> + >R $@ ;

: $"                           (S  --  )
       ASCII "  STATE @
       IF    COMPILE ["] 0 C,
             WORD C@ -1 ALLOT DUP , ALLOT
       ELSE  0 C, WORD C@ -1 ALLOT HERE !
             HERE DUP 2+ SWAP @ $@
       THEN ;                            IMMEDIATE

: $IN                          (S n --  )
       PAD DUP ROT EXPECT SPAN @ $@ ;

    22
\ $VARIABLE $VAR-ARRAY                               05/10/85ck

: $var_build                   (S $len -- $len )
       DUP , 0 ,  DUP HERE SWAP BLANK  DUP ALLOT ;

: $VARIABLE
       CREATE                  (S compile:  $len --  )
             $var_build DROP
       DOES>   2+ ;            (S     run:    -- addr )

: $VAR-ARRAY
       CREATE                  (S compile: $len size --  )
             0 DO  $var_build
               LOOP DROP
       DOES>                   (S     run:   n -- addr )
             SWAP OVER @ 4 + * + 2+ ;

    23
\ $VAR@ $VAR! $VARFILL                               05/13/85ck

: $VARFILL                     (S addr c --  )
       OVER 2- @ ROT 2+ SWAP ROT FILL ;

: $VAR@                        (S addr --  )
       DUP 2+ SWAP @ $@ ;

: $VAR!                        (S addr --  )
       DUP >R  DUP BL $VARFILL
       DUP 2+ SWAP 2- @ ?1$P@ @ MIN  DUP >R
       $P@ 2+ ROT ROT MOVE
       R> R> !
       $DROP ;
```

```
    72
                                                     05/07/85ck
["]                            (S  --  )
  Moves in-line string to $stack.

$"                             (S  --  )
  If compiling emplace an in-line string to be moved to string
  stack at execution time, else put enclosed string on string
  stack. Used in the form:   $" File not found"

$IN                            (S n --  )
  A simple input line editor that get a string of maximum
  length n and leaves it on the $string. This format uses
  F83's version of EXPECT which can be redefined to change the
  functions of the keys by changing the execution array
  pointed to by the variable CC.

    73
                                                     03/26/85ck
$VARIABLE                      (S compile:  $len --  )
                               (S     run:    -- addr )
  Used in the form 15 $VARIABLE <name> to create a new string
  variable <name> with space for 15 characters. When <name>
  is executed it returns the address of the length (16 bits)
  of the currently stored string. The maximum length of a
  string variable's buffer is stored at addr-2.
$VAR-ARRAY                     (S compile: $len size --  )
                               (S     run:   n -- addr )
  Used in the form 15 8 $VARIABLE <name> to create a new array
  of string variables <name> with space for 8 strings of 15
  characters each. When <name> is executed it converts the
  element number n to the address of the length (16 bits)
  of the currently stored string. The maximum length of a
  string array variable's buffer is stored at addr-2.

    74
                                                     03/26/85ck
$VARFILL                       (S addr c --  )
  Fill the $variable at address with the character c.

$VAR@                          (S addr --  )
  Fetch the string at address and push it on the $stack. Only
  the character originally saved will be returned. If the
  $variable is empty, a null string will be returned.

$VAR!                          (S addr --  )
  Pop the top string from the $stack and save it in the
  $variable at address. The actual string length is saved with
  the characters for later use. If the top string is too long
  it will be trunctated on the right.
```

```
    24
\ $CONST-ARRAY                                           05/10/85ck

: $CONST-ARRAY
      CREATE                  (S compile:  $len --    )
              C, ASCII " WORD  C@ 1+ ALLOT
      DOES>                   (S      run:   n --    )
              DUP C@  DUP >R
              ROT * + 2+  R>  $@ ;

: $CONSTANT
      CREATE                  (S compile:    --    )
              ASCII " WORD  C@ 1+ ALLOT
      DOES>                   (S      run:    --    )
              DUP 1+  SWAP C@  $@ ;
```

```
    75
                                                          05/10/85ck
$CONST-ARRAY                  (S compile:  $len --    )
                              (S      run:   n --    )
   Create & access an initialized array of string constants.
   Because WORD returns an 8 bit length, the maximum length
   of a string constant array is 256 characters.
   NOTE: use only one blank after the array name.
      example:   6 $CONST-ARRAY NAME "Cliff Janet LaurenKent  "
      where:     1 NAME would put 'Janet ' on $stack.

$CONSTANT                     (S compile:    --    )
                              (S      run:    --    )
   Create & access an initialized string constant.
   NOTE: use only one blank after the $CONSTANT's name.
      example:   $CONSTANT NAME "Cliff"
      where:     NAME would put 'Cliff' on $stack.
```

```
    25
\ number to string conversion                            03/26/85ck

: $str          OVER - BL $CHRS $@ ;

\ stack for following words:    num field --
: $D.L  >R (D.) R> $str $+ ;
: $D.R  >R (D.) R> $str $SWAP $+ ;
: $N.L  >R (.)  R> $str $+ ;
: $N.R  >R (.)  R> $str $SWAP $+ ;

\ stack for following words:    num  --
: $D.   (D.) $@ ;
: $U.   (U.) $@ ;
: $N.   (.)  $@ ;
```

```
    76
                                                          03/26/85ck
These words parallel the standard FORTH number printing words.
Each creates a string on the $stack. If the field width will not
contain the number the string will be too long; no data is lost.

Stack for following words:    d field --
   $D.L  32 bit left justified
   $D.R  32 bit right justified
Stack for following word:     d --
   $D.   32 bit signed
Stack for following words:    n field --
   $N.L  16 bit left justified
   $N.R  16 bit right justified
Stack for following words:    n --
   $U.   16 bit un-signed
   $N.   16 bit signed
```

```
    26
\ $DOLLARS                                               05/10/85ck

: (decimal$D.R)               (S d-num field places --    )
      SWAP >R >R  TUCK DABS <#
      R> 0
      ?DO  #  LOOP  ASCII . HOLD
      #S ROT SIGN #>
      R> OVER - BL $CHRS  $@ $SWAP $+ ;

: $DOLLARS                    (S d-num field --    )
      1-  2  (decimal$D.R)
      1 ASCII $ $CHRS $+ ;
```

```
    77
                                                          03/26/85ck
(decimal$D.R)                 (S d-num field places --    )
   Convert a double number to a right justified string with
   'field' characters and 'places' digits after the decimal.


$DOLLARS                      (S d-num field --    )
   Using field width at TOS, convert the d-number / 100 to a
   string as dollars and cents. Note that the dollar sign and
   the decimal point are included in the character count, so
   that there are 2 digits less than the field width. If the
   field width will not contain the number, the string will be
   too long; no data will be lost.
```

27

```
\ $LC->UC  $UC->LC                                05/10/85ck

: $LC->UC                         (S  --  )
        Top$ UPPER ;

: $UC->LC                         (S  --  )
        Top$
        OVER + SWAP
        ?DO  I C@ DUP
             ASCII A ASCII Z BETWEEN
             IF     32 + I C!
             ELSE   DROP
             THEN
        LOOP ;
```

78

```
                                                   03/26/85ck
$LC->UC                            (S  --  )
  Replace all lower case with upper case.


$UC->LC                            (S  --  )
  Replace all upper case with lower case.
```

28

```
\ $TRIM $STRIP                                     03/26/85ck

: $TRIM                           (S  --  )
        Top$ -TRAILING
        $@  $SWAP  $DROP ;

: $STRIP                          (S  --  )
        Top$ SWAP
        OVER 0
        ?DO     DUP C@ BL =
                IF    1+ SWAP 1- SWAP
                ELSE  LEAVE
                THEN
        LOOP
        SWAP $@ $SWAP $DROP ;
```

79

```
                                                   03/26/85ck
$TRIM                              (S  --  )
  Remove trailing blanks from top string. This is the $stack
  equivalant of -TRAILING.


$STRIP                             (S  --  )
  Remove leading blanks from top string.
```

29

```
\ $VAL                                             05/13/85ck

: $VAL                            (S  -- d f )
        $STRIP  $LC->UC
        $P@ @  PAD >R
        DUP 2+ R@ SWAP BLANK
        R@ C!
        R@ 1+ $!
        R> NUMBER? ;
```

80

```
                                                   03/26/85ck
$VAL                               (S  -- d f )
  Converts the top string to a double number, using the current
  system base. The string is destroyed.  A leading minus sign
  is allowed. Leading and trailing blanks are also allowed,
  however no blanks are allowed between a minus sign and
  the number that follows. The system variable DPL will
  contain the number of characters to the right of the decimal,
  if any. The flag at TOS indicates the success or failure
  of the conversion.
```

```
    30
\ $=                                                03/26/85ck

: $=                              (S  -- f )
        Top$
        Sec$
        ROT MIN
        COMPARE ;
```

```
    81
                                                    03/26/85ck

$=                                (S  -- f )
   Compare the two top strings, to the length of the shorter
   string. The flag may take any of three values:
         0 - the strings are equal
         1 - the top string is less that the second
        -1 - the top string is greater than the second
   Neither string is lost or altered.
```

```
    35
\ $stack inspection                                 06/12/85ck
HEX
: CLR$S
        $P0 100 - 100 ERASE ASCII * $P0 C! ;

: .$S   $P0  F0 - 100 DUMP
        CR ." Current top: "
        BASE @   HEX  $P@ U. ." hex "
        BASE ! ;

: .$V                             (S addr --   )
        10 -  50 DUMP ;
DECIMAL
\S
```

```
    86
                                                    06/12/85ck

CLR$S clears the top 256 bytes of the $stack to zeros to make
   debuging with .$S easier.

.$S is a non destructive dump of the top 256 bytes of the string
   stack area in hex format. This will show string contents,
   string order and the string lengths.

.$V                               (S addr --   )
   Displays a string variable in memory.
```

```
     0
                                                    05/07/85ck


                    TOOLS.BLK

        Extentions to the Laxen & Perry F83 Model

                 1985 Clifford Kent

             KENT ENGINEERING & DESIGN
                   P.O. Box 178
                 Mottville NY 13119
                  (315)685-8237
```


```
     0
                                                    05/07/85ck


                    TOOLS.BLK

        Extentions to the Laxen & Perry F83 Model

                 1985 Clifford Kent

             KENT ENGINEERING & DESIGN
                   P.O. Box 178
                 Mottville NY 13119
                  (315)685-8237
```

U.S.

• ALABAMA

Huntsville FIG Chapter
Call Tom Konantz
205/881-6483

• ALASKA

Kodiak Area Chapter
Call Horace Simmons
907/486-5049

• ARIZONA

Phoenix Chapter
Call Dennis L. Wilson
602/956-7678

Tucson Chapter
Twice Monthly,
2nd & 4th Sun., 2 p.m.
Flexible Hybrid Systems
2030 E. Broadway #206
Call John C. Mead
602/323-9763

• ARKANSAS

Central Arkansas Chapter
Twice Monthly, 2nd Sat., 2p.m. &
4th Wed., 7 p.m.
Call Gary Smith
501/227-7817

• CALIFORNIA

Los Angeles Chapter
Monthly, 4th Sat., 10 a.m.
Hawthorne Public Library
12700 S. Grevillea Ave.
Call Phillip Wasson
213/649-1428

Monterey/Salinas Chapter
Call Bud Devins
408/633-3253

Orange County Chapter
Monthly, 4th Wed., 7 p.m.
Fullerton Savings
Talbert & Brookhurst

Fountain Valley
Monthly, 1st Wed., 7 p.m.
Mercury Savings
Beach Blvd. & Eddington
Huntington Beach
Call Noshir Jesung
714/842-3032

San Diego Chapter
Weekly, Thurs., 12 noon
Call Guy Kelly
619/268-3100 ext. 4784

Sacramento Chapter
Monthly, 4th Wed., 7 p.m.
1798-59th St., Room A
Call Tom Ghormley
916/444-7775

Bay Area Chapter
Silicon Valley Chapter
Monthly, 4th Sat.
FORML 10 a.m., Fig 1 p.m.
H-P Auditorium
Wolfe Rd. & Pruneridge,
Cupertino
Call John Hall 415/532-1115
or call the FIG Hotline:
408/277-0668

Stockton Chapter
Call Doug Dillon
209/931-2448

• COLORADO

Denver Chapter
Monthly, 1st Mon., 7 p.m.
Call Steven Sarns
303/477-5955

• CONNECTICUT

Central Connecticut Chapter
Call Charles Krajewski
203/344-9996

• FLORIDA

Orlando Chapter
Every two weeks, Wed., 8 p.m.
Call Herman B. Gibson
305/855-4790

Southeast Florida Chapter
Monthly, Thurs., p.m.
Coconut Grove area
Call John Forsberg
305/252-0108

Tampa Bay Chapter
Monthly, 1st. Wed., p.m.
Call Terry McNay
813/725-1245

• GEORGIA

Atlanta Chapter
3rd Tuesday each month, 6:30 p.m.
Computone Cottilion Road
Call Ron Skelton
404/393-8764

• ILLINOIS

Cache Forth Chapter
Call Clyde W. Phillips, Jr.
Oak Park
312/386-3147

Central Illinois Chapter
Urbana
Call Sidney Bowhill
217/333-4150

Fox Valley Chapter
Call Samuel J. Cook
312/879-3242

Rockwell Chicago Chapter
Call Gerard Kusiolek
312/885-8092

• INDIANA

Central Indiana Chapter
Monthly, 3rd Sat., 10 a.m.
Call John Oglesby
317/353-3929

Fort Wayne Chapter
Monthly, 2nd Wed., 7 p.m.
Indiana/Purdue Univ. Campus
Rm. B71, Neff Hall
Call Blair MacDermid
219/749-2042

• IOWA

Iowa City Chapter
Monthly, 4th Tues.
Engineering Bldg., Rm. 2128
University of Iowa
Call Robert Benedict
319/337-7853

Central Iowa FIG Chapter
Call Rodrick A. Eldridge
515/294-5659

Fairfield FIG Chapter
Monthly, 4th day, 8:15 p.m.
Call Gurdy Leete
515/472-7077

• KANSAS

Wichita Chapter (FIGPAC)
Monthly, 3rd Wed., 7 p.m.
Wilbur E. Walker Co.
532 Market
Wichita, KS
Call Arne Flones
316/267-8852

• LOUISIANA

New Orleans Chapter
Call Darryl C. Olivier
504/899-8922

• MASSACHUSETTS

Boston Chapter
Monthly, 1st Wed.
Mitre Corp. Cafeteria
Bedford, MA
Call Bob Demrow
617/688-5661 after 7 p.m.

• MICHIGAN

Detroit Chapter
Monthly, 4th Wed.
Call Tom Chrapkiewicz
313/562-8506

• MINNESOTA

MNFIG Chapter
Even Month, 1st Mon., 7:30 p.m.
Odd Month, 1st Sat., 9:30 a.m.
Vincent Hall Univ. of MN
Minneapolis, MN
Call Fred Olson
612/588-9532

• MISSOURI

Kansas City Chapter
Monthly, 4th Tues., 7 p.m.
Midwest Research Institute
MAG Conference Center
Call Linus Orth
913/236-9189

St. Louis Chapter
Monthly, 1st Tues., 7 p.m.
Thornhill Branch Library
Contact Robert Washam
91 Weis Dr.
Ellisville, MO 63011

• NEVADA

Southern Nevada Chapter
Call Gerald Hasty
702/452-3368

• NEW HAMPSHIRE

New Hampshire Chapter
Monthly, 1st Mon., 6 p.m.
Armtec Industries
Shepard Dr., Grenier Field
Manchester
Call M. Peschke
603/774-7762

• NEW MEXICO

Albuquerque Chapter
Monthly, 1st Thurs., 7:30 p.m.
Physics & Astronomy Bldg.
Univ. of New Mexico
John Bryon
Call 505/298-3292

• NEW YORK

FIG, New York
Monthly, 2nd Wed., 8 p.m.
Queens College
Call Ron Martinez
212/517-9429

Rochester Chapter
Bi-Monthly, 4th Sat., 2 p.m.
Hutchinson Hall
Univ. of Rochester
Call Thea Martin
716/235-0168

Rockland County Chapter
Call Elizabeth Gormley
Pearl River
914/735-8967

Syracuse Chapter
Monthly, 3rd Wed., 7 p.m.
Call Henry J. Fay
315/446-4600

• OHIO

Akron Chapter
Call Thomas Franks
216/336-3167

Athens Chapter
Call Isreal Urieli
614/594-3731

Cleveland Chapter
Call Gary Bergstrom
216/247-2492

Cincinatti Chapter
Call Douglas Bennett
513/831-0142

Dayton Chapter
Twice monthly, 2nd Tues., &
4th Wed., 6:30 p.m.
CFC 11 W. Monument Ave.
Suite 612

Dayton, OH
Call Gary M. Granger
513/849-1483

• **OKLAHOMA**

Central Oklahoma Chapter
Monthly, 3rd Wed., 7:30 p.m.
Health Tech. Bldg., OSU Tech.
Call Larry Somers
2410 N.W. 49th
Oklahoma City, OK 73112

• **OREGON**

Greater Oregon Chapter
Monthly, 2nd Sat., 1 p.m.
Tektronix Industrial Park
Bldg. 50, Beaverton
Call Tom Almy
503/692-2811

• **PENNSYLVANIA**

Philadelphia Chapter
Monthly, 4th Sat., 10 a.m.
Drexel University, Stratton Hall
Call Melanie Hoag or Simon Edkins
215/895-2628

• **TENNESSEE**

East Tennessee Chapter
Monthly, 2nd Tue., 7:30 p.m.
Sci. Appl. Int'l. Corp., 8th Fl.
800 Oak Ridge Turnpike, Oak Ridge
Call Richard Secrist
615/483-7242

• **TEXAS**

Austin Chapter
Contact Matt Lawrence
P.O. Box 180409
Austin, TX 78718

Dallas/Ft. Worth
Metroplex Chapter
Monthly, 4th Thurs., 7 p.m.
Call Chuck Durrett
214/245-1064

Houston Chapter
Call Dr. Joseph Baldwin
713/749-2120

Periman Basin Chapter
Call Carl Bryson
Odessa
915/337-8994

• **UTAH**

North Orem FIG Chapter
Contact Ron Tanner
748 N. 1340 W.
Orem, UT 84057

• **VERMONT**

Vermont Chapter
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Rm. 210, Monkton Rd.
Vergennes, VT
Call Don VanSyckel
802/388-6698

• **VIRGINIA**

First Forth of Hampton Roads
Call William Edmonds
804/898-4099

Potomac Chapter
Monthly, 2nd Tues., 7 p.m.
Lee Center
Lee Highway at Lexington St.
Arlington, VA
Call Joel Shprentz
703/860-9260

Richmond Forth Group
Monthly, 2nd Wed., 7 p.m.
154 Business School
Univ. of Richmond
Call Donald A. Full
804/739-3623

• **WISCONSIN**

Lake Superior FIG Chapter
Monthly, 2nd Fri., 7:30 p.m.
University of Wisconsin
Superior
Call Allen Anway
715/394-8360

Milwaukee Area Chapter
Call Donald H. Kimes
414/377-0708

MAD Apple Chapter
Contact Bill Horzon
129 S. Yellowstone
Madison, WI 53705

FOREIGN

• **AUSTRALIA**

Melbourne Chapter
Monthly, 1st Fri., 8 p.m.
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
03/29-2600

Sydney Chapter
Monthly, 2nd Fri., 7 p.m.
John Goodsell Bldg.
Rm. LG19
Univ. of New South Wales
Sydney
Contact Peter Tregeagle
10 Binda Rd., Yowie Bay
02/524-7490

• **BELGIUM**

Belgium Chapter
Monthly, 4th Wed., 20:00h
Contact Luk Van Loock
Lariksdreff 20
2120 Schoten
03/658-6343

Southern Belgium FIG Chapter
Contact Jean-Marc Bertinchamps
Rue N. Monnom, 2
B-6290 Nalinnes
Belgium
071/213858

• **CANADA**

Alberta Chapter
Call Tony Van Muyden
403/962-2203

Nova Scotia Chapter
Contact Howard Harawitz
227 Ridge Valley Rd.
Halifax, Nova Scotia B3P2E5
902/477-3665

Southern Ontario Chapter
Quarterly, 1st Sat., 2 p.m.
General Sciences Bldg., Rm. 312
McMaster University
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S4K1
416/525-9140 ext. 3443

Toronto FIG Chapter
Contact John Clark Smith
P.O. Box 230, Station H
Toronto, ON M4C5J2

• **COLOMBIA**

Colombia Chapter
Contact Luis Javier Parra B.
Aptdo. Aereo 100394
Bogota
214-0345

• **ENGLAND**

Forth Interest Group — U.K.
Monthly, 1st Thurs.,
7p.m., Rm. 408
Polytechnic of South Bank
Borough Rd., London
D.J. Neale
58 Woodland Way
Morden, Surry SM4 4DS

• **FRANCE**

French Language Chapter
Contact Jean-Daniel Dodin
77 Rue du Cagire
31100 Toulouse
(16-61)44.03.06

• **GERMANY**

Hamburg FIG Chapter
Monthly, 4th Sat., 1500h
Contact Horst-Gunter Lynsche
Common Interface Alpha
Schanzenstrasse 27
2000 Hamburg 6

• **HOLLAND**

Holland Chapter
Contact: Adriaan van Roosmalen
Heusden Houtsestraat 134
4817 We Breda
31 76 713104

FIG des Alpes Chapter
Contact: Georges Seibel
19 Rue des Hirondelles
74000Annely
50 57 0280

• **IRELAND**

Irish Chapter
Contact Hugh Doggs
Newton School
Waterford
051/75757 or 051/74124

• **ITALY**

FIG Italia
Contact Marco Tausel
Via Gerolamo Forni 48
20161 Milano
02/645-8688

• **JAPAN**

Japan Chapter
Contact Toshi Inoue
Dept. of Mineral Dev. Eng.
University of Tokyo
7-3-1 Hongo, Bunkyo 113
812-2111 ext. 7073

• **NORWAY**

Bergen Chapter
Kjell Birger Faeraas
Hallskaret 28
Ulset
+47-5-187784

• **REPUBLIC OF CHINA**

R.O.C.
Contact Ching-Tang Tzeng
P.O. Box 28
Lung-Tan, Taiwan 325

• **SWEDEN**

Swedish Chapter
Hans Lindstrom
Gothenburg
+46-31-166794

• **SWITZERLAND**

Swiss Chapter
Contact Max Hugelshofer
ERNI & Co., Elektro-Industrie
Stationsstrasse
8306 Bruttisellen
01/833-3333

SPECIAL GROUPS

Apple Corps Forth Users Chapter
Twice Monthly, 1st &
3rd Tues., 7:30 p.m.
1515 Sloat Boulevard, #2
San Francisco, CA
Call Robert Dudley Ackerman
415/626-6295

Baton Rouge Atari Chapter
Call Chris Zielewski
504/292-1910

FIGGRAPH
Call Howard Pearlmutter
408/425-8700

FORTH INTEREST GROUP
P. O. Box 8231
San Jose, CA 95155

BULK RATE
U.S. POSTAGE
PAID
Permit No. 3107
San Jose, CA

Address Correction Requested