A USER'S GUIDE TO "FOCAL" FOR THE 6502

"FOCAL" IS A REGISTERED TRADEMARK OF
DIGITAL EQUIPMENT CORPORATION

COPYRIGHT 1977 THE DENVER 6502 USER'S GROUP

DISTRIBUTED BY: THE 6502 PROGRAM EXCHANGE
2920 WEST HOANA LANE
RENO, NEVADA 89509
"FOCAL" IS THE NAME GIVEN TO A HIGH-LEVEL MATHEMATICAL LANGUAGE INTERPRETER ORIGINALLY CONCEIVED FOR THE DIGITAL EQUIPMENT CORPORATION PDP-8 SERIES OF MINI-COMPUTERS. "FOCAL" HAS HISTORICALLY BEEN A LANGUAGE FOR BEGINNERS (A LA "BASIC") AND A LANGUAGE USED BY THE EXPERIENCED HACKER. THIS MANUAL DESCRIBES "FOCAL" AS IT EXISTS ON THE 6502 MICROPROCESSOR.

THIS USER’S GUIDE IS PRESENTED IN A "LET’S TAKE A GUIDED TOUR OF FOCAL" FORMAT. READERS ARE ENCOURAGED TO PROVIDE CONSTRUCTIVE FEEDBACK CONCERNING THIS MANUAL, WHICH WAS PRODUCED USING "FOCAL" ON A 6502 MICROPROCESSOR
TABLE OF CONTENTS

"TYPE" COMMAND, IMMEDIATE NUMERICAL AND STRING OUTPUT ........................................ 1
"SET" COMMAND, VARIABLES, SUBSCRIPTS .............................................................................. 8
FORMATTING OUTPUT FOR "TYPE", MULTIPLE-STATEMENT LINES ........................................ 17
PROGRAM STRUCTURE: GROUPS AND STEPS, "WRITE," "GOTO" AND "ERASE" COMMANDS .... 22
EDITING PROGRAMS, "MODIFY" COMMAND ........................................................................ 27
DATA INPUT: "ASK" COMMAND, FORMATTING, ERRORS ......................................................... 30
PROGRAM CONTROL: "IF" COMMAND .................................................................................. 35
CALLING GROUPS WITH "DO" COMMAND; "QUIT" AND "RETURN"; "ON" .......................... 44
LOOPS: "IF" STRUCTURES, "FOR" COMMAND .................................................................. 58
FUNCTIONS: ABSOLUTE VALUE, TAKE INTEGER, RANDOM VALUE ................................. 66
FUNCTIONS: ASCII CHARACTER INPUT AND OUTPUT .................................................... 71
FUNCTIONS: DIRECTED INPUT AND OUTPUT .................................................................. 74
STRINGS: LITERALS, FUNCTIONS, I/O, STRINGS AS DEVICES ........................................ 78
FUNCTIONS: USER-DEFINED SUBROUTINES, SOFTWARE INTERRUPT ............................ 93
FUNCTIONS: CONSOLE ECHO, CURSOR CONTROL, MEMORY EXAMINE/DEPOSIT ............ 100
APPENDIX A: ERROR CODES ............................................................................................... 101
APPENDIX B: TRIGONOMETRY/MATHEMATICAL SUBROUTINES ........................................... 102
APPENDIX C: MODIFYING AND AUGMENTING FOCAL-65 .................................................. 103
COMMANDS AND FUNCTIONS

A ASK A X A "TEXT",X;

C COMMENT 2.35 C THIS STEP WILL NOT EXECUTE

D DO D 10 D 22.35

E ERASE E E 12 E 22.35 E ALL

F FOR F I=1,10;T "LOOP",! F I=2,2,10;T "EVENS",I,!

G GO G G 2.75

I IF I (X) 2.1, 2.2, 2.3 I (Y-X), 3.1; T "> THAN",!

M MODIFY M 1.55

O ON ON (X) 1, 2, 3

Q QUIT Q

R RETURN R

S SET S X=23+(2*B) S X=FRAN(1)

T TYPE T A,B,C T X-Y T 3*C T "HELLORLD",!

W WRITE W W 3 W 6.2

FABS ABSOLUTE VALUE FRAN RANDOM RUMBER 0-.99999

FINT RETURN INTEGER FINR RETURN ROUNDED INTEGER

FCHR INPUT ASCII CHAR. FOUT OUTPUT ASCII CHAR.

FMEM 'PEEK' OR 'POKE' FSBR USER SUBROUTINES

FECH CONSOLE ECHO CTRL FPIC INTERRUPT SERVICE

FIDV SET INPUT DEVICE FODV SET OUTPUT DEVICE

FSL SET STRING LENGTH FSLK COMPARE STRINGS

FSTI READ CHARS. FROM INPUT FSTO SEND STRING TO OUTPUT

FINI INITIALIZE INPUT DEVICE FINO INITIALIZE OUTPUT DEVICE

FCON SET CONSOLE FCUR SET CURSOR
FOCAL MUST BE GIVEN 'COMMANDS' IN ORDER TO ACTUALLY ACCOMPLISH
SOMETHING USEFUL TO THE USER. THESE COMMANDS INSTRUCT FOCAL TO
PERFORM A SPECIFIC OPERATION OR SERIES OF OPERATIONS. THE FOCAL
'SYSTEM', WHICH RESIDES IN THE COMPUTER'S MEMORY, HAS BEEN
DESIGNED TO UNDERSTAND A SPECIFIC SET OF COMMANDS. ANY COMMAND
THAT YOU GIVE TO FOCAL MUST BE ONE OF THESE SPECIFIC COMMANDS
THAT IT HAS DESIGNED TO RECOGNIZE, IF YOU TRY TO GIVE
FOCAL OTHER COMMANDS, IT WILL NOT KNOW HOW TO INTERPRET THEM.

ONE OF THE MOST USEFUL COMMANDS IS THE 'TYPE' COMMAND, THE
'TYPE' COMMAND ALLOWS THE USER TO GIVE FOCAL AN ARITHMETIC
EXPRESSION, HAVE FOCAL EVALUATE IT, AND TYPE THE RESULTANT VALUE
ON THE USER'S OUTPUT DEVICE. SOME SIMPLE EXAMPLES FOLLOW:

*TYPE 1+1
2.000*

LET US LOOK AT THE ABOVE 'TYPE' COMMAND. THE USER ENTERS THE
COMMAND BY TYPING 'TYPE 1+1', AND THEN STRIKING THE 'CARRIAGE RETURN'
KEY ON HIS KEYBOARD. THIS KEY IS SOMETIMES LABELED AS
'RETURN'. FOCAL DOES NOTHING WITH THE COMMAND UNTIL THE 'RETURN'
KEY IS STRUCK. AT THAT POINT, FOCAL THEN TRIES TO INTERPRET THE
COMMAND AS ONE OF THOSE THAT IT HAS BEEN DESIGNED TO RECOGNIZE
(SUCH AS 'TYPE') AND THEN DOES THE APPROPRIATE THING THAT THE
COMMAND INDICATES TO DO; IN THIS CASE, FOCAL WAS TOLD TO EVALUATE
THE ARITHMETIC EXPRESSION '1+1' AND TYPE THE RESULTANT VALUE TO
THE OUTPUT DEVICE. IT DID THIS, SINCE THE VALUE 2.000* APPEARED
ON THE OUTPUT DEVICE. AT THAT POINT, FOCAL HAD ACCOMPLISHED EVERYTHING
THAT THE COMMAND INDICATED, SO IT OUTPUTS THE '#' CHARACTER, WHICH
IS A PROMPT, TELLING THE USER THAT IT HAS NOWHERE MORE TO DO.
FOCAL THEN WAITS FOR THE USER TO ENTER A NEW COMMAND. A 'BLANK'
(SPACE BAR) MUST ALWAYS FOLLOW A FOCAL COMMAND, SINCE THIS IS
USED TO SEPARATE THE COMMAND NAME (SUCH AS 'TYPE') FROM THE REST
OF THE INFORMATION ON THE LINE (SUCH AS '1+1'). HOWEVER, THE COMMAND
NAME (SUCH AS 'TYPE') NEEDS ONLY TO BE EXPRESSED AS A SINGLE
CHARACTER (IN THIS CASE 'T') IN ORDER FOR FOCAL TO UNDERSTAND
WHICH COMMAND IT IS. SOME MORE EXAMPLES FOLLOW:

*T 1+1
2.000*
*T 2+3
5.000*
*T 2*2+4
8.000*
AS YOU CAN SEE. IF I JUST STRIKE THE 'RETURN' KEY, WITHOUT
TYING A COMMAND OF SOME KIND, THEN FOCAL HAS NOTHING TO DO, AND
SIMPLY PROMPTS AGAIN WITH THE '"' LIKE SOME MORE EXAMPLES OF THE
'"TYPE' COMMAND"

```
12.5+3.2
15.750

10-7.5
2.500

3*4
12.000

3/4
2.750

2*2*2
8.000

2*3
8.000
```

AS YOU CAN SEE, SEVERAL DIFFERENT ARITHMETIC OPERATIONS CAN BE
PERFORMED BY FOCAL. THESE ARE ADDITION, SUBTRACTION, DIVISION, MULTI-
PLICATION, AND EXPONENTIATION (RAISING TO A POWER). THESE OPERATIONS
ARE INDICATED BY THE SYMBOLS +, -, /, *, **, RESPECTIVELY.
WHEN THESE OPERATIONS ARE MIXED WITHIN A GIVEN EXPRESSION, THERE
IS A HIERARCHY RULE WHICH FOCAL USES TO DETERMINE THE ORDER IN WHICH
IT IS TO PERFORM THE OPERATIONS. FOCAL WILL PERFORM ANY EXPONENTIATION
FIRST (**), THEN ANY MULTIPLICATION (*), THEN ANY DIVISION (/), THEN
ANY SUBTRACTION (-), AND FINALLY ANY ADDITIONS (+). SOME EXAMPLES
ILLUSTRATING THIS RULE FOLLOW:

```
1+3*4
13.000

2-3/4
1.250

3-2
1.000

2-3
-1.000

-2-3/4
-2.750

3*5/2
7.500

3*5/2+2
9.500

24/3*4
2.000
```
THE USER MAY INDICATE THAT A CERTAIN GROUP OF OPERATIONS IS
TO BE PERFORMED FIRST. HE DOES THIS BY ENCLOSING THAT GROUP OF
OPERATIONS WITHIN PARENTHESES. THERE MAY BE MORE GROUPS ENCLOSED
WITH PARENTHESES WHICH ARE CONTAINED WITHIN A GROUP ALREADY ENCLOSED
WITH PARENTHESES. IN THIS CASE, FOCUS WILL PERFORM THE
OPERATIONS WITHIN THE MOST DEEPLY NESTED (INNERMOST) PARENTHESES
FIRST, THEN THOSE IN THE NEXT OUTER, AND SO ON, UNTIL THOSE AT THE
OUTERMOST LEVEL ARE DONE LAST. HERE ARE SOME EXAMPLES TO CLARIFY
THIS RULE:

1. 1+1
   2.000

2. (1+1)
   2.000

3. 24/3*4
   2.000

4. (24/3)*4
   32.000

5. 1+(2*(3+4))
   15.000

6. 2*3
   8.000

7. 2*(1-3)
   0.125

8. 1/8
   0.125

9. 1+(2*(3+4)+3*(4/2))
   21.500

MESSAGE MAY ALSO BE OUTPUT BY USING THE 'TYPE' COMMAND.
BY SIMPLY ENCLOSING A SERIES OF CHARACTERS INSIDE OF DOUBLE
QUOTATION MARKS ('"'), THE 'TYPE' COMMAND WILL OUTPUT THE
SERIES OF CHARACTERS JUST AS THEY APPEAR INSIDE THE QUOTATION
MARKS. SOME EXAMPLES:

"HI THERE"
"HI THERE"

"NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY"
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY

"ANY SERIES OF CHARACTERS"
ANY SERIES OF CHARACTERS
THE USER MAY INSTRUCT THE 'TYPE' COMMAND TO PERFORM SEVERAL
FUNCTIONS BY SEPARATING EACH FUNCTION FROM THE NEXT WITH
A COMMA (,). SOME EXAMPLES FOLLOW:

T 1+1,2/3,4+2
  2.000 0.667  16.000

T "THE ANSWER IS",2+2
THE ANSWER IS  4.000

T "FIRST ANSWER IS",3-2,"SECOND ANSWER IS",5/2+1
FIRST ANSWER IS  1.000 SECOND ANSWER IS  3.900

AS YOU CAN SEE, THIS CAPABILITY ALLOWS OUTPUT FROM THE
COMPUTER TO BE MADE MORE LEGIBLE. SOMETIMES IT IS
DESIRED TO HAVE CONTROL OVER THE LINE SPACING ON THE OUTPUT
DEVICE, IN ORDER TO MAKE THE OUTPUT APPEAR MORE LEGIBLE. THERE
ARE SPECIAL FORMAT CONTROL CHARACTERS WHICH FOCAL RECOGNIZES WHEN
THEY APPEAR IN A 'TYPE' STATEMENT WHICH ALLOW THE USER TO DO THIS
KIND OF FORMATTING. ONE SUCH CHARACTER IS THE EXCLAMATION MARK (!).
WHEN FOCAL ENCOUNTERS THIS CHARACTER IN A 'TYPE' STATEMENT, IT OUTPUTS
A CARRIAGE RETURN CHARACTER, FOLLOWED BY A LINE FEED CHARACTER, TO
THE OUTPUT DEVICE. THIS CAUSES A RETURN TO THE BEGINNING OF THE
CURRENT LINE, AND AN ADVANCING TO THE NEXT LINE ON THE OUTPUT
DEVICE. THE POUND CHARACTER (#), WHEN ENCOUNTERED IN A 'TYPE'
STATEMENT, CAUSES A CARRIAGE RETURN CHARACTER TO BE OUTPUT, BUT
THAT'S ALL. THE EFFECT IS THAT THE CARRIAGE IS RETURNED
BUT THE LINE IS NOT ADVANCED. HENCE, ANY FURTHER OUTPUT WOULD OCCUR
ON THE SAME LINE, POSSIBLY OVERPRINTING EXISTING OUTPUT. SOME
EXAMPLES FOLLOW:

T 1=1,
  2.000

T 1=1,1/2+3,!
  2.000  8.000

T "THE ANSWER IS ",3+4,1,"THE VALUE OF THE DISTANCE IS",4+3,!
THE ANSWER IS 12.000
THE VALUE OF THE DISTANCE IS 64.000

T 1=1,1,2+3,11
  2.000  8.000

T "Hi",1," THERE",11
HI THERE

T "Hi",8," THERE",11
HI THERE

T !!!," X  Y","1,2+3,4/5,11

X  Y
  5.000  0.800
THIS CAPABILITY ALLOWS FOR MAKING VERY READABLE OUTPUT.

SOMETIMES IT IS USEFUL TO HAVE FOCAL REMEMBER THE RESULT OF AN
ARITHMETIC CALCULATION. THIS RESULT MAY THEN BE USED LATER INSTEAD.
OF HAVING TO RE-DO THE CALCULATION OVER AGAIN. THIS CAPABILITY IS
ACCOMPLISHED THROUGH THE USE OF 'VARIABLE NAMES'. A NAME
MAY BE ATTACHED TO THE RESULT, OR PARTIAL RESULT, OF AN ARITHMETIC
CALCULATION, IF THE NAME IS USED LATER IN SOME OTHER CALCULATION.
THEN THE VALUE ASSOCIATED WITH THAT NAME IS SUBSTITUTED IN PLACE OF
THE NAME, IN FOCAL, A VARIABLE NAME MUST BEGIN WITH A LETTER OF
THE ALPHABET (A-Z), BUT MAY NOT BEGIN WITH THE LETTER 'F' (THIS LETTER
HAS A SPECIFIC SIGNIFICANCE, DISCUSSED LATER). A VARIABLE NAME MAY ALSO
HAVE AN OPTIONAL SECOND CHARACTER, WHICH MUST BE A DIGIT IN THE RANGE
OF 0-7. WHEN THE OPTIONAL DIGIT IS OMITTED, THEN '0' IS
ASSUMED AS THE DIGIT. THUS, 'A3', 'B7', 'C1', 'B2', AND 'R' ARE
VALID NAMES, WHILE SUCH NAMES AS, 'F2', 'B81', AND 'AB' ARE NOT VALID.
NAMES FOR VARIABLE QUANTITIES, THE TERM 'VARIABLE' IS USED. BECAUSE,
AT SOME LATER TIME, THE SAME NAME MAY BE ASSOCIATED WITH A NEW QUANTITY,
THUS A NAME'S VALUE MAY 'VARY', FROM TIME TO TIME. SOME EXAMPLES
WILL HELP TO CLARIFY THIS.

1 = 1.1
2.000

x = 1.1
2.000

y = 1.1
2.000

x = 2.000
3.000

x = 3.000

x = 3.000

y = 3.000

z = 5.000

y = 5.000

x = 5.000

5.000

x = 3.000

y = 5.000

x = 1.1

y = 1.1

x = 4.000

y = 6.000

x = 4.000

y = 6.000

1 = 1.1

2.000
X, Y, 1
4.200  6.000

A1=x=1,1
5.000

A1=x, y, 1
5.000  4.000  6.000

"A1=x, A1, 1, x, =, y, =, Y, 1, !

A1= 5.000
X= 4.000
Y= 6.000

"A1=x, A1, 1, x, =, y, =, Y, 1, !

A1= 5.000
X= 4.000
Y= 6.000

"C IN ORDER TO MAKE THE ABOVE MORE PRETTY
"T "A1=x, A1, 1, x, =, y, =, Y, 1, !

A1= 5.000
X= 4.000
Y= 6.000

"T "A1=x, A1, 1, x, =, y, =, Y, 1, !
AS YOU CAN SEE, THE VALUES CURRENTLY ASSIGNED TO THE NAMES
'A', 'X', AND 'Y' WILL BE RETAINED BY FOCAL FOR USE IN LATER EXPRESSIONS.
ALSO NOTE THAT THE TWO NAMES 'A' AND 'AB' ARE ONE AND THE SAME.

SOME MORE EXAMPLES:

A = 3 + 2.1
9.200
A = 9.200
A = A + 1
10.000
A = A + 1
10.000 10.000
2 = (A + 1)
22.000
2 = A + 1
21.000
2 = A - 1
19.000
A = A
10.000
2 = (X * A + 1)
22.000
A = A
10.000 11.000
2 = (A + A - 1)
22.000
A = A
11.000
A = A
11.000
A = X, y, A
5.000 11.000 6.000 11.000
A = X + Y, A
2.000
A = X, y, A
2.000 2.000 2.000 2.000
*C ANY TIME A PARTIAL EXPRESSION NEEDS TO BE REMEMBERED, JUST
*C SET A VARIABLE NAME EQUAL TO THE PARTIAL EXPRESSION. REMEMBER THAT
*C IN ALL CASES, FOCAL CAN BE FORCED TO PERFORM THE APPROPRIATE
*C EXPRESSION EVALUATION AND NAME SUBSTITUTION, THROUGH THE PROPER
*C USE OF PARENTHESES.

*C IF THE USER WANTS TO EVALUATE AN EXPRESSION, BUT DOES NOT WANT
*C THE RESULT TYPED OUT, THEN WE MAY USE THE FOCAL COMMAND 'SET'. THIS
*C FOCAL COMMAND PERFORMS ANY VARIABLE NAME SUBSTITUTIONS, JUST LIKE THE
*C 'TYPE' COMMAND, BUT DOES NOT OUTPUT. SOME EXAMPLES OF THE 'SET' COMMAND:

*SET X=1.5
*C NOTICE THAT NO OUTPUT IS DONE, HOWEVER!
*T X.
* 1.500

*C THE 'SET' COMMAND CAN BE ABBREVIATED TO A SINGLE LETTER 'S'.
*C (NOTE: ALL FOCAL COMMANDS CAN BE ABBREVIATED TO A SINGLE LETTER).
*C SOME MORE EXAMPLES OF 'SET':
*S X=1+1+1
*T X.
* 3.000
*S X=1,Y=2,Z=3
*T X,Y,Z.
* 1.000 2.000 3.000
*T X=1+1
* 2.000
*T X,Y,Z.
* 8.000 2.000 3.000
*S A=3+1.5
*T A,X,Y,Z, & A.
* 9.500 8.000 2.000 3.000 2.000
*S 1+1
*T A,X,Y,Z, & A.
* 9.500 8.000 2.000 3.000 2.000

*C NOTE: SINCE NO SUBSTITUTION TOOK PLACE IN THE 'S 1+1' COMMAND ABOVE,
*C THEN THE COMMAND SIMPLY EVALUATED THE EXPRESSION '1+1' AND DID
*C NOTHING WITH THE RESULT!
- PAGE 9 -

C A FEW MORE EXAMPLES:

S A = X

T A1, X, Y, Z, A1;
  9.500  8.000  2.000  3.000  6.000

S A = Y = Z = A1

T A1, X, Y, Z, A1;

C VARIABLE NAMES (SUCH AS 'A1', 'X', 'Y', 'Z', 'A1') CAN HAVE 'SUBSCRIPTS'
C ASSOCIATED WITH THEM. A SUBSCRIPT IS ESSENTIALLY AN 'ITEM' OR 'ELEMENT'.
C NUMBER WHICH MAY FURTHER DEFINE THE VARIABLE. FOCAL SUBSCRIPTS ARE
C ROUGHLY ANALOGOUS TO THE MATHEMATICAL SUBSCRIPTS USED IN ALGEBRA. 
C IS USEFUL, SOMETIMES, TO DEAL WITH SPECIFIC ITEMS OF A GIVEN NAME.
C FOR EXAMPLE, IF THE NAME 'C1' REPRESENTED THE CHAIRS IN A ROOM, THEN
C 'C(2)' (IE, SUBSCRIPT '2') MIGHT REPRESENT THE SECOND CHAIR IN THE ROOM.
C 'C(3)' MIGHT REPRESENT THE NEXT, 'C(2)' MIGHT REPRESENT THE
C NEXT, AND 'C(n)' MIGHT REPRESENT THE 'nTH' CHAIR. IF THERE WERE AT
C LEAST 'n' CHAIRS IN THE ROOM, SUBSCRIPTS IN FOCAL MAY BE ANY
C VALUE IN THE RANGE OF -32769 TO +32767. SOME EXAMPLES SHOULD HELP
C TO CLARIFY SUBSCRIPTS!

S X(1) = 5

T X(1): 1
  5.000

T X:
  8.000

C NOTE THAT X AND X(1) ARE DIFFERENT. IN FACT, IF A SUBSCRIPT IS
C OMITTED, THE VALUE OF 0 IS ASSUMED. THUS X AND X(0) ARE ONE AND
C THE SAME, MORE EXAMPLES!

T X(0):
  8.000

S X = X + 1

T X, X(1):
  9.000  9.000
A useful option in the 'TYPE' command is the 'ST' option, when a 'ST' is encountered in a 'TYPE' command, FOCAL prints all of the variable names, their subscripts, and their current value.

More examples:

```
> TYPE

X(0) = 9.000
Y(0) = 9.500
A(0) = 9.500
Z(0) = 9.500
X(1) = 5.000

> !"My variables and their values are!":.1,$.11

My variables and their values are:

X(0) = 9.000
Y(0) = 9.500
A(0) = 9.500
Z(0) = 9.500
X(1) = 5.000

> !"Would it become evident that X and X(0) are the same? Actually, X, X(0), and X(0) are all one and the same variable name.

More examples:

> S X(0)=10,X(1)=9,X(2)=8,X(3)=7

> TYPE

X(0) = 10.000
Y(0) = 9.500
A(0) = 9.500
Z(0) = 9.500
X(1) = 9.000
X(2) = 8.000
X(3) = 7.000
```
C THE ORDER IN WHICH THE VARIABLES APPEAR WHEN PRINTED WITH THE 'S'
C OPTION OF 'TYPE' IS THE ORDER IN WHICH THEY WERE FIRST GIVEN VALUES,
C IF A VARIABLE NAME IS USED IN A FOCAL STATEMENT WITHOUT HAVING BEEN
C PREVIOUSLY GIVEN A VALUE, IT IS DEFINED AT THAT POINT AND GIVEN THE
C VALUE OF ZERO.  THE 'ERASE' COMMAND, ABBREVIATED 'E' CAN BE
C USED TO ERASE ALL DEFINED VARIABLES AND THEIR VALUES, HENCE IMPLICITLY
C DEFINING ALL THE VALUES TO BE ZERO.  (MORE ON THE 'ERASE' COMMAND LATER).
C
C MORE EXAMPLES:

C
C
C ERASE
C
C S

C NOTE THAT THERE ARE NOW NO DEFINED VARIABLE NAMES;
C
C SET X(0)=10,X(1)=9,X(2)=8,X(3)=7,X(4)=6,X(5)=5,X(6)=4,X(7)=3,X(8)=2,X(9)=1
C
C S

X0( 0)= 10.000
X0( 1)= 9.000
X0( 2)= 8.000
X0( 3)= 7.000
X0( 4)= 6.000
X0( 5)= 5.000
X0( 6)= 4.000
X0( 7)= 3.000
X0( 8)= 2.000
X0( 9)= 1.000
S
S

X0( 0)= 10.000
X0( 1)= 9.000
X0( 2)= 8.000
X0( 3)= 7.000
X0( 4)= 6.000
X0( 5)= 5.000
X0( 6)= 4.000
X0( 7)= 3.000
X0( 8)= 2.000
X0( 9)= 1.000
N=5
S
*S N=N-3
*TS
X0(0) = 18,000
X0(1) = 9,000
X0(2) = 9,000
X0(3) = 7,000
X0(4) = 6,000
X0(5) = 5,000
X0(6) = 4,000
X0(7) = 3,000
X0(8) = 2,000
X0(9) = 1,000
N0(0) = 2,000

*T X(N),1
  8,000
*T X(2),1
  8,000
*T X(N+1),1
  7,000
*T X(N+1),1
  9,000
*T X(N-1),1
  9,000
*TS
X0(0) = 18,000
X0(1) = 9,000
X0(2) = 9,000
X0(3) = 7,000
X0(4) = 6,000
X0(5) = 5,000
X0(6) = 4,000
X0(7) = 3,000
X0(8) = 2,000
X0(9) = 1,000
N0(0) = 2,000
*S X(9)=X(6)*1
*T $ 
X( 0)= 10.000
X( 1)= 9.000 
X( 2)= 8.000 
X( 3)= 7.000 
X( 4)= 6.000 
X( 5)= 5.000 
X( 6)= 4.000 
X( 7)= 3.000 
X( 8)= 2.000 
X( 9)= 1.000 
N( 0)= 2.000 

*S X(9)*N=1
*T $ 
X( 0)= 10.000
X( 1)= 9.000 
X( 2)= 8.000 
X( 3)= 7.000 
X( 4)= 6.000 
X( 5)= 5.000 
X( 6)= 4.000 
X( 7)= 3.000 
X( 8)= 2.000 
X( 9)= 1.000 
N( 0)= 2.000 

*C OK. WATCH THIS CAREFULLY!
*T X(N),! 
   8.000 
   *T X(X(N)), 
      2.000 
   *T X(3),! 
      2.000
AN ARITHMETIC EXPRESSION IN FOCAL IS DIRECTLY REDUCIBLE TO
A SINGLE NUMBER, THUS, A SUBSCRIPT CAN BE A CONSTANT (SUCH AS '2'),
OR A VARIABLE (SUCH AS 'X' OR 'B(N)'), OR AN ENTIRE
EXPRESSION (SUCH AS '1+2*X'), SINCE ALL OF THESE ARE DIRECTLY
REDUCIBLE TO A SINGLE NUMBER, IF THAT SINGLE NUMBER IS WITHIN
THE RANGE OF -32,767 TO +32,767, THEN IT MAY BE USED AS A SUBSCRIPT,
FOCAL TRUNCATES (DROPS) ANY FRACTIONAL PART OF THE NUMBER BEFORE IT
USES IT AS A SUBSCRIPT.

MORE EXAMPLES:

```
SET J(1000) = 20
SET S
X(0)= 10.000
X(1)= 9.000
X(2)= 8.000
X(3)= 7.000
X(4)= 6.000
X(5)= 5.000
X(6)= 4.000
X(7)= 3.000
X(8)= 2.000
X(9)= 1.000
N2(0)= 2.000
J0(1000)= 20.000
SET J(2000) = 40
SET S
X(0)= 10.000
X(1)= 9.000
X(2)= 8.000
X(3)= 7.000
X(4)= 6.000
X(5)= 5.000
X(6)= 4.000
X(7)= 3.000
X(8)= 2.000
X(9)= 1.000
N2(0)= 2.000
J0(1000)= 20.000
J0(2000)= 40.000
```
*C IT IS NOT NECESSARY (AS IN SOME OTHER COMPUTER LANGUAGES) TO
*C DEFINE THE INTERVENING SUBSCRIPTED VALUES, IN ORDER TO USE A
*C PARTICULAR VALUE.  THUS, FOCAL ARRAYS (VARIABLES WITH SUBSCRIPTS)
*C ARE CONSIDERED TO BE 'SPARSE'.
*C MORE EXAMPLES:

*E
*T S

*S K (1-10)=1, K (1-9)=2, K (1-1)=9, K (1-4000)=2222, K=3
*T S

K0 (1-10)= 1.000
K0 (1-9)= 2.000
K0 (1-1)= 9.000
K0 (1-4000)= 2222.000
K0 (0)= 3.000

*S K1 (1-10)=1, K1 (1-9)=2, K1 (1-1)=9, K1 (1-4000)=2222, K1=3
*T S

K1 (1-10)= 1.000
K1 (1-9)= 2.000
K1 (1-1)= 9.000
K1 (1-4000)= 2222.000
K1 (0)= 3.000
*C THIS SHOWS THAT THE VARIABLE NAMES 'K' AND 'KB' ARE THE SAME, BUT
*C ARE DIFFERENT FROM 'K1'.
*C MORE EXAMPLES:

*E
*T S
*S N=4
*T S
N3(0) = 4,000

T S
N3(0) = 4,000
N0(0) = 10,000

*T B(1), B(2), B(3), 1
 0.200  0.000  0.000

*T S
N3(0) = 4,000
N0(0) = 10,000
B0(1) = 0,000
B0(2) = 0,000
B0(3) = 0,000

*  
*C THIS SHOWS THAT IF A VARIABLE NAME HAS NOT BEEN PREVIOUSLY ASSIGNED
*C A VALUE, THAT IT IS ASSIGNED THE VALUE OF ZERO THE FIRST TIME THAT
*C THE USER REFERS TO THE NAME.
NOTICE THAT WHEN NUMBERS ARE OUTPUT BY FOCAL, THEY ARE OUTPUT IN A CERTAIN FORMAT. SOMETIMES THE USER WOULD LIKE TO CHANGE THE FORMAT IN WHICH FOCAL OUTPUTS NUMBERS. THIS IS ACCOMPLISHED USING THE 'X' (PERCENT) OPTION OF THE 'TYPE' COMMAND. THIS OPTION ONLY ESTABLISHES THE FORMAT THAT SUBSEQUENT NUMBERS WILL BE OUTPUT IN, BUT DOES NO ACTUAL OUTPUT ITSELF. ONCE THE OUTPUT FORMAT IS SET, ALL SUBSEQUENT NUMBERS WILL BE OUTPUT IN THAT FORMAT UNTIL THE FORMAT IS CHANGED WITH ANOTHER 'X' OPTION TO THE 'TYPE' COMMAND. THE 'X' OPTION MAY BE INTERSPEORED WITH OTHER OPTIONS OF THE 'TYPE' COMMAND JUST LIKE '3', '1', AND '9'. WHEN THE 'X' IS ENCOUNTERED IN A 'TYPE' COMMAND, A NUMBER OF THE FORM 'BB,AA' IS ASSUMED TO FOLLOW IT. THE 'BB' (00-99) INDICATES HOW MANY DIGITS ARE TO BE OUTPUT BEFORE THE DECIMAL POINT (THE INTEGER PORTION OF THE NUMBER OUTPUT). THE 'AA' (00-99) INDICATES HOW MANY DIGITS ARE TO BE OUTPUT FOLLOWING THE DECIMAL POINT (THE FRACTIONAL PORTION OF THE NUMBER OUTPUT). IF 'AA' IS ZERO, THEN NO DECIMAL POINT IS ACTUALLY OUTPUT. THE ACTUAL NUMBER OF CHARACTERS OUTPUT DOES NOT INCLUDE THE DECIMAL POINT, NOR THE '+' SIGN PRECEDING THE NUMBER (IF ITS NEGATIVE). NOTE THAT '95.03' MEANS FIVE DIGITS BEFORE THE DECIMAL POINT, AND THREE DIGITS AFTER THE DECIMAL POINT. '95.3' OR '95.30' MEANS FIVE DIGITS BEFORE AND THREE DIGITS AFTER THE DECIMAL POINT. THE NUMBERS USED IN ACTUAL CALCULATIONS ARE MAINTAINED TO ABOUT 6 DIGITS OF SIGNIFICANCE. THEY MAY LIE WITHIN THE RANGE OF 10Y-38 TO 10Y+38. SOME EXAMPLES SHOULD HELP CLARIFY THE 'X' OPTION TO THE 'TYPE' COMMAND.
-- PAGE 18 --

* NOTICE THAT FOCAL WILL ALWAYS OUTPUT THE INTEGER PORTION OF THE
* NUMBER, EVEN THOUGH IT MIGHT NOT BE ABLE TO FIT IN THE NUMBER OF
* DIGITS THE USER HAS ASKED FOCAL TO PLACE BEFORE THE DECIMAL POINT.
* THIS AT LEAST ALLOWS THE USER TO SEE THE NUMBER, EVEN THOUGH THE
* DECIMAL POINTS WILL NO LONGER Line UP IN COLUMN DATA.
* MORE EXAMPLES:

* T 6.5, !
 12.000

* T 25.03

* C NO OUTPUT WAS DONE IN THIS CASE, BUT A NEW OUTPUT FORMAT WAS
* C SPECIFIED, MORE EXAMPLES:

* T 6.5, !
 12.000

* T 2.345, !
 2.345

* T 2.3456, !
 2.346

* C NOTICE THAT THERE WAS MORE FRACTIONAL PART IN THE NUMBER THAN THE
* C OUTPUT FORMAT SPECIFIED TO OUTPUT, IN THIS CASE FOCAL WILL ROUND
* C BEFORE OUTPUTTING THE VALUE; NOTE THAT THE ACTUAL VALUE OF THE
* C NUMBER THAT FOCAL HAS STORED INTERNALLY HAS NOT BEEN CHANGED.
* C THE ROUNDOING WAS ONLY DONE FOR THE OUTPUT OPERATION, MORE EXAMPLES

* SET X=2.3456

* T X,
 2.346

* T X-.04, X,
 2.3456

* T X-.02, X,
 2.35

* T X-.01, X,
 2.3

* T X-.0, X,
 2

* T X-.04, X,
 2.3456
S X=2.78
T X,!
    2.78
T %5.02,X,!
    2.78
T %5.01,X,!
    2.8
T %5.00,X,!
    3.0
S X=4
T %5.03,X,!
    4.000
T %5.3,X,!
    4.00000000000000000000000000000000
T %5.03,X,!
    4.000

IT IS NOT NECESSARY FOR THE VALUE FOLLOWING THE 'X' TO BE A CONSTANT
(SUCH AS '5.03'), BUT CAN BE ANY ARITHMETIC EXPRESSION. FOCAL
WILL EVALUATE THE EXPRESSION, REDUCE IT TO A NUMBER OF THE FORM
'00.00' AND USE THAT VALUE AS THE FORMAT SPECIFIER. SOME EXAMPLES:
S X=3.02
T X,!
    3.020

CURRENT OUTPUT FORMAT IS STILL '5.03' FROM ABOVE.
T 1+1,!
    2.000
T XX,1+1,!
    2.00

THE OUTPUT FORMAT IS NOW '3.02', BECAUSE THAT IS THE VALUE OF 'X'.
MORE EXAMPLES:
S X=X+1
T XX,1+1,!
    2.00
* T S

XG( 0) =  4.02

* THE OUTPUT FORMAT IS NOW '4.02', BECAUSE THAT IS THE VALUE OF 'X'.
* WATCH THIS ONE CAREFULLY!
* T X*X+0.01,1+1,!
  2.000
* T S

XG( 0) =  4.030

* THE OUTPUT FORMAT WAS SET TO 'X' (AFTER 'X' WAS INCREMENTED BY .01).
* SO THE OUTPUT FORMAT IS NOW '4.03', WHICH MEANS THAT FOUR DIGITS ARE OUTPUT
* BEFORE THE DECIMAL POINT, AND THREE DIGITS ARE OUTPUT AFTER THE DECIMAL
* POINT. MORE EXAMPLES!

* T 1
  1.000*
* T 1,1,2,1,3,!
  1.000
  2.000
  3.000

* T -1,1,-2,1,-3,!
 -1.000
 -2.000
 -3.000

* T X*X+0.03,1+1,!
  2.000
LET'S NOW LOOK AT AN IMPORTANT FEATURE OF FOCAL, THE 'TYPE' COMMANDS ALLOW THE USER TO DO SOME USEFUL THINGS. THE USER MAY NEED TO DO A SEQUENCE OF 'TYPE' AND 'SET' COMMANDS IN ORDER TO ACCOMPLISH A CERTAIN TASK, THE USER CAN PLACE MORE THAN ONE COMMAND PER LINE BY SEPARATING EACH COMMAND WITH A ';' (SEMICOLON).

S X=5, Y=6, Z=7; T X, Y, Z;

S X=5, Y=6, Z=7; T X, Y, Z;

S X=5, Y=6, Z=7; T X, Y, Z;

S X=5, Y=6, Z=7; T X, Y, Z;

S X=5, Y=6, Z=7; T X, Y, Z;

S X=5, Y=6, Z=7; T X, Y, Z;
ONE OF THE MOST USEFUL FEATURES OF ANY COMPUTER LANGUAGE IS THE
ABILITY TO STORE A SERIES OF COMMANDS FOR LATER EXECUTION. THE
COMPUTER IS VERY GOOD AT EXECUTING A SEQUENCE OF INSTRUCTIONS (COMMANDS)
OVER AND OVER AGAIN. THE USER MAY STORE A LINE OF FOCAL COMMANDS
AWAY FOR LATER EXECUTION. THIS IS DONE BY TYPING A 'LINE NUMBER'
BEFORE THE ACTUAL LINE OF COMMANDS. WHEN THE CARTRIDGE RETURN IS
STRIKED, THE LINE WILL BE STORED BY FOCAL, BUT THE COMMANDS ON THE
LINE WILL NOT BE EXECUTED UNTIL OPPOSED TO IMMEDIATE EXECUTION IF THE
'LINE NUMBER' IS LEFT OFF, 'LINE NUMBERS' IN FOCAL ARE COMPOSED OF
TWO PARTS, AND HAVE THE FORM 'GG,SS', WHERE 'GG' (01-99) IS
THE 'GROUP' THAT THIS LINE BELONGS TO, AND 'SS' (01-99) IS THE
'Step' WITHIN THE 'GROUP'. WE WILL SEE THE SIGNIFICANCE OF BEING
ABLE TO 'GROUP' LINES TOGETHER AND REFER TO THEM AS A UNIT, THUS A
'LINE NUMBER' OF '2,35' INDICATES THAT THIS LINE IS 'STEP' NUMBER 35
IN 'GROUP' 2, NOTE THAT '2,2' FOR A LINE NUMBER MEANS THAT THIS LINE
IS STEP TWENTY IN GROUP TWO, AND NOT STEP TWO. '2,09' WOULD BE
USED TO INDICATE THAT THE LINE WAS STEP TWO IN GROUP TWO. SOME
EXAMPLES ARE IN ORDER HERE:

1.1 T "HELLO, THERE"!;; $X=1,Y=2,Z=3
1.2 T "THE VALUES OF X, Y, Z ARE $X,Y,Z!

2.1 T "THAT'S ALL FOLKS",!

NOTICE THAT THE COMMANDS ON THESE LINES WERE NOT EXECUTED, BUT THE
LINES WERE STORED AWAY BY FOCAL, SO THAT WE MAY EXECUTE THEM ANY
TIME WE DESIRE. WHenever A STORED LINE IS ENTERED (OR CHANGED), THE
USER'S VARIABLE NAMES AND THEIR VALUES ARE ERASED FROM THE COMPUTER'S
STORAGE. THERE ARE SEVERAL FOCAL COMMANDS WHICH ARE USEFUL
TO THE USER BECAUSE THEY ALLOW MANIPULATION OF STORED LINES. WE WILL
BE LOOKING AT THEM INDIVIDUALLY AS NEEDED. THE FIRST OF THESE COMMANDS
IS THE 'WRITE' COMMAND ("W") WHICH ALLOWS THE USER TO WRITE OUT
THE LINES (ALL OR SOME) THAT FOCAL HAS STORED. EXAMPLE:

WRITE
C FOCAL=65 (V3D) 18-JUL-77
1.10 T "HELLO, THERE"!;; $X=1,Y=2,Z=3
1.20 T "THE VALUES OF X, Y, Z ARE $X,Y,Z!
2.10 T "THAT'S ALL FOLKS",!

W
C FOCAL=65 (V3D) 18-JUL-77
1.10 T "HELLO, THERE"!;; $X=1,Y=2,Z=3
1.20 T "THE VALUES OF X, Y, Z ARE $X,Y,Z!
2.10 T "THAT'S ALL FOLKS",!
*C IF NO PARAMETER FOLLOWS THE 'W' OR 'WRITE' COMMAND, THEN ALL LINES
C WHICH FOCAL HAS STORED AWAY WILL BE WRITTEN TO THE OUTPUT DEVICE.
C ALSO WHENEVER FOCAL WRITES 'ALL' THE LINES, IT WRITES THE TOP LINE
C WHICH IS AN IDENTIFIER TELLING WHICH VERSION OF THE FOCAL SYSTEM
C THIS IS, AND THE DATE WHICH IT HAS CREATED. IF A SPECIFIC LINE
C NUMBER FOLLOWS THE 'W' OR 'WRITE' COMMAND, THEN ONLY THAT LINE
C IS WRITTEN TO THE OUTPUT DEVICE. EXAMPLE:

*W 1.2
1.20 T "THE VALUES OF X, Y, Z, ARE "",X,Y,Z,!

*W 2.1
2.10 T "THAT'S ALL FOLKS!"!

*W 1.1
1.10 T "HELLO, THERE"!!IS X=1,Y=2,Z=3

*W 1.1W 1.21W 1.1
1.10 T "THAT'S ALL FOLKS!"!

1.20 T "THE VALUES OF X, Y, Z; ARE "",X,Y,Z;!

1.10 T "HELLO, THERE"!!IS X=1,Y=2,Z=3

*T : "HERE ARE MY STORED LINES",!!IW
HERE ARE MY STORED LINES

C FOCAL=65 (V3D) 18=JULY=79
1.10 T "HELLO, THERE"!!IS X=1,Y=2,Z=3.
1.20 T "THE VALUES OF X, Y, Z, ARE "",X,Y,Z,!
2.10 T "THAT'S ALL FOLKS!"!

*C THE USEFULNESS OF 'GROUPS' OF LINES WILL NOW BECOME APPARENT.
*C IF THE USER SPECIFIES ONLY A GROUP NUMBER WITHOUT A LINE NUMBER
*C (LINE NUMBER OF ZERO), THEN THE 'WRITE' COMMAND WRITES OUT ALL LINES
*C WHICH BELONG TO THE SPECIFIED GROUP. EXAMPLE:

*WRITE 1
1.10 T "HELLO, THERE"!!IS X=1,Y=2,Z=3
1.20 T "THE VALUES OF X, Y, Z; ARE "",X,Y,Z;!

*W 2
2.10 T "THAT'S ALL FOLKS!"!
THIS ALLOWS THE USER TO LIST ANY LINE, GROUP, OR THE ENTIRE
PROGRAM. A 'PROGRAM' IS A SERIES OF STORED LINES WHICH PERFORM SOME
FUNCTION OR TASK FOR THE USER. WELL, WE HAVE CAREFULLY TYPED IN THE
ABOVE STORED PROGRAM, BUT HOW DO WE INSTRUCT FOCAL TO ACTUALLY
PERFORM THE COMMANDS THAT HAVE BEEN STORED AWAY? WE CAN INDICATE THAT
FOCAL TRANSFER CONTROL (BEGIN EXECUTING STATEMENTS) TO ANY SPECIFIC
LINE BY THE USE OF THE 'GOTO' (ABBREVIATED 'GO' OR 'IGO') COMMAND.
THE 'GOTO' COMMAND MUST BE FOLLOWED WITH THE LINE NUMBER OF
THE STORED LINE THAT WE WANT TO TRANSFER CONTROL TO, IF A LINE NUMBER
IS OMITTED, THEN CONTROL IS TRANSFERRED TO THE LOWEST NUMBERED
LINE THAT HAS BEEN STORED AWAY. SOME EXAMPLES:

```
1 20  T "HELLO, THERE" : IJS X=1,Y=2,Z=3
1 20  T "THE VALUES OF X, Y, Z ARE 1, 2, 3";
2 10  T "THAT'S ALL FOLKS";

GOTO 1.1
HELLO, THERE
THE VALUES OF X, Y, Z ARE 1.000 2.000 3.000
THAT'S ALL FOLKS

* GOTO 1.2
THE VALUES OF X, Y, Z ARE 1.000 2.000 3.000
THAT'S ALL FOLKS

* GOTO 2.1
THE VALUES OF X, Y, Z ARE 1.000 2.000 3.000
THAT'S ALL FOLKS

* S X=11,Y=12,Z=13 IJS 1.2
THE VALUES OF X, Y, Z ARE 11.000 12.000 13.000
THAT'S ALL FOLKS

* GOTO
HELLO, THERE
THE VALUES OF X, Y, Z ARE 1.000 2.000 3.000
THAT'S ALL FOLKS

* FOCAL=65 (Y3D) 18-JUL-77
```
C FOCAL=69 (V3O) 18-JUL-77

1.10 T "HELLO, THERE"! IS X#1; Y#2; Z#3
1.15 G 2.1
1.20 T "THE VALUES OF X, Y, Z ARE N,X,Y,Z!,

2.10 T "THAT'S ALL FOLKS!"

#GO
HELLO, THERE
THAT'S ALL FOLKS

# NOTE THAT THE NORMAL SEQUENTIAL DIRECTION OF STATEMENT EXECUTION
# WAS ALTERED WHEN THE LINE '1.15' WAS INSERTED. WE WERE INTRODUCED TO
# THE 'ERASE' COMMAND EARLIER. HOWEVER, THERE ARE OTHER USES FOR THE
# 'ERASE' COMMAND. IF THE ERASE COMMAND IS FOLLOWED BY A SPECIFIC
# LINE NUMBER (SUCH AS '1.21'), IT ERASES JUST THAT LINE FROM STORAGE.
# IF IT IS FOLLOWED BY A GROUP NUMBER ONLY (SUCH AS '1'), THEN IT ERASES
# ALL LINES THAT BELONG TO THAT GROUP. IF IT IS FOLLOWED BY THE
# WORD 'ALL', THEN IT ERASES ALL OF THE STORED LINES. AS WE
# HAVE BEEN BEFORE, IF THE 'ERASE' COMMAND IS FOLLOWED BY NO LINE OR
# GROUP NUMBER, THEN IT ONLY ERASES THE USERS VARIABLE NAMES AND THEIR STORED
# VALUES, ALL FORMS OF THE 'ERASE' COMMAND REMOVE ANY VARIABLE NAMES
# AND THEIR ASSIGNED VALUES. SOME EXAMPLES:

# E 1.15
#

C FOCAL=69 (V3O) 18-JUL-77

1.10 T "HELLO, THERE"! IS X#1; Y#2; Z#3
1.20 T "THE VALUES OF X, Y, Z ARE N,X,Y,Z!

2.10 T "THAT'S ALL FOLKS!"

# 2.2 T "THIS IS THE LIVING END!" !
C FOCAL=65 (V3D) 18-JUL-77

1.10 T "HELLO, THERE!! IS X=1,Y=2,Z=3"
1.20 T "THE VALUES OF X, Y, Z ARE 1,2,3"

2.10 T "THAT'S ALL FOLKS!!!
2.20 T "THIS IS THE LIVING END!!!

*Go
HELLO, THERE

THE VALUES OF X, Y, Z ARE 1,000, 2,000, 3,000
THAT'S ALL FOLKS
THIS IS THE LIVING END!!
* E 2
*W

C FOCAL=65 (V3D) 18-JUL-77

1.10 T "HELLO, THERE!! IS X=1,Y=2,Z=3"
1.20 T "THE VALUES OF X, Y, Z ARE 1,2,3"

*Go
HELLO, THERE

THE VALUES OF X, Y, Z ARE 1,000, 2,000, 3,000
* E ALL
*W

C FOCAL=65 (V3D) 18-JUL-77

*Go
SINCE THERE WERE NO STORED LINES, THERE WAS NOTHING FOR FOCAL
TO PERFORM. AT THIS POINT, IT WOULD BE APPROPRIATE TO DISCUSS THE
QUESTION "WHAT IF I MAKE A MISTAKE?". FOCAL PROVIDES SEVERAL
MECHANISMS WHICH ASSIST THE USER IN CORRECTING MISTAKES. SOME WILL
BE DISCUSSED HERE; OTHERS LATER. "WHAT IF I MAKE A MISTAKE WHILE
TYPING COMMANDS TO FOCAL FROM THE KEYBOARD?". SINGLE CHARACTERS
CAN BE 'RUBBED OUT' BY STRIKING THE 'RUBOUT' KEY (SOMETIMES LABELED
'DELETE!') ON THE KEYBOARD. ONE CHARACTER TO THE LEFT WILL BE
'FORGOTTEN' BY FOCAL FOR EACH TIME THE 'RUBOUT' KEY IS STRUCK,
SOME EXAMPLES:

"THIS IS A TEST"

NOTICE THAT THE '\' CHARACTER IS ECHOED EVERY TIME THE 'RUBOUT' KEY
IS STRUCK (A FANCY RUBOUT MODE FOR CRTS IS AVAILABLE, WHERE THE
CHARACTER IS 'EATEN' OFF THE SCREEN). THE ENTIRE LINE TO THE
LEFT CAN BE 'FORGOTTEN' BY STRIKING THE 'BACKARROW' KEY ON THE KEYBOARD,
AN EXAMPLE:

"THIS IS A TEST"

WHEN THE 'BACKARROW' KEY WAS STRUCK, FOCAL FORGOT EVERYTHING TO THE
LEFT OF THE 'BACKARROW'; THE USER MAY JUST CONTINUE TYPING THE NEW
CHARACTERS ON THE SAME LINE. MORE EXAMPLES:

"THIS ARE\" IS A TEST"

HERE SEVERAL 'RUBOUTS' AND 'BACKWARDS' WERE USED ON THE SAME
LINE. (YOU WOULD NEVER MAKE THAT MANY MISTAKES, OF COURSE).
"WHAT IF A MISTAKE IS FOUND INSIDE OF A LINE ALREADY STORED AWAY?"
ONE APPROACH WOULD BE TO SIMPLY TYPE THE WHOLE LINE IN OVER AGAIN (FOCAL
WILL REPLACE THE ONE STORED WITH THE NEW LINE), BUT THIS PROCESS CAN
BE VERY TEDIOUS IF ONLY ONE CHARACTER IS TO BE CHANGED IN A LONG LINE,

FOCAL PROVIDES A FACILITY WITH THE 'MODIFY' COMMAND TO ALLOW THE
USER TO MAKE CHANGES IN A STORED LINE WITHOUT HAVING TO RE-TYPE THE
ENTIRE LINE. IF THE USER TYPES THE 'MODIFY' COMMAND ('M'), FOLLOWED
BY A SPECIFIC LINE NUMBER, THAT LINE IS OPENED FOR MODIFICATION. FOCAL
TYPES OUT THE LINE NUMBER (INFORMING THE USER THAT IT LOCATED THE LINE),
THEN WAITS FOR USER INPUT. WHENEVER THE MODIFY COMMAND IS WAITING FOR
USER INPUT, THE USER HAS SEVERAL MODIFICATION OPTIONS. THEY ARE

1. SIMPLY TYPE IN ANY TEXT THAT THE USER WANTS INSERTED AT THAT POINT;
2. DELETE ANY TEXT TO THE LEFT BY THE USE OF THE 'TRUBOUT' AND/OR
   'BACKARROW' KEYS;
3. SEARCH FOR (POSITION AFTER) A CHARACTER FURTHER TO THE RIGHT OF THE
   USER'S CURRENT POSITION ON THE LINE. THIS IS DONE BY STRIKING
   THE 'ALT-LOC' KEY (SOMETIMES LABELED 'ESCAPE'), FOLLOWED BY
   STRIKING THE CHARACTER THAT IS TO BE SEARCHED FOR. IF THE CHARACTER
   IS LOCATED, THE LINE IS TYPED OUT UP TO THAT POINT AND FOCAL
   WAITS FOR FURTHER USER INPUT. IF THE CHARACTER IS NOT LOCATED,
   THE ENTIRE REST OF THE LINE IS TYPED, AND MODIFICATION ENDS;
4. THE USER MAY TRUNCATE A LINE (REMOVE ALL INFORMATION TO THE RIGHT)
   BY TYPING THE CARRIAGE RETURN KEY. AT THIS POINT MODIFICATION ENDS.
5. THE USER MAY END MODIFICATION BY STRIKING THE 'LINE FEED' KEY;
   THIS CAUSES THE REMAINDER OF THE LINE TO BE TYPED, AND THEN
   MODIFICATION ENDS. ALL DEFINED VARIABLE NAMES AND THEIR VALUES
   ARE ALSO ERASED.
* IF THE USER WISHES TO MODIFY THE LINE AGAIN, HE SIMPLY TYPES A NEW
* MODIFY COMMAND. THIS PROCESS IS A LITTLE HARD TO DEMONSTRATE ON THE
* PRINTED PAGE, BUT HERE ARE SOME EXAMPLES:
*
* C FOCAL=65 (V3D) 18-JUL-77
* 1.1 T "THIS IS MY PROGRAM",!
*
* C FOCAL=65 (V3D) 18-JUL-77
* 1.10 T "THIS IS MY PROGRAM",!
* DO
* THIS IS MY PROGRAM
* H 1.1
* 1.10 T "THIS IS MY PROGRAM",!
* DO
* THIS IS MY PROGRAM
* H
*
* C FOCAL=65 (V3D) 18-JUL-77
* 1.10 T "THIS IS MY PROGRAM",!
*
* WHAT THE USER DID, IN THE ABOVE MODIFY COMMAND, WAS TO TYPE AN
* ALT MODE, THEN THE CHARACTER 'R'. FOCAL THEN TYPED OUT THE LINE UP
* TO THE NEXT 'R' ENCOUNTERED (THE 'R' IN 'PROGRAM'). THE USER THEN
* STRUCK THE 'D' KEY, WHICH JUST INSERTED AN 'O' AT THAT POINT; THEN
* STRUCK THE 'LINE FEED' KEY IN ORDER TO RETAIN THE REMAINDER OF THE LINE
* AS IS. THAT REQUIRED 4 KEYSTROKES AS OPPOSED TO 26 REQUIRED TO RE-TYPE
* THE ENTIRE LINE.
Sometimes it is useful for the user to supply data values for some of the variables in the program. Run the program, then supply different values, run the program again, etc. The user can input numerical information from the input device and have focal store that information in a variable name, just as if he had used the 'set' command to do it. The focal command which does this is the 'ask' (or 'at') command.

For each variable name appearing in the 'ask' command, focal waits for an arithmetic expression to be input, reduces it to a single numeric value, and assigns that value as the value of the variable name.

Example:

C focal=65 (v30) 18-jul-77

$ ask x,y,z
12.34
27.312
5
$ X,Y,Z,!!
22.340 27.312 5.000

$ x0(0)= 12.340
y0(0)= 27.312
z0(0)= 5.000

$ x,y,z
1.5,3

$ x0(0)= 1.000
y0(0)= 2.000
z0(0)= 3.000

$ X,Y,Z,!!
1.000 2.000 3.000
Certain non-numeric characters can separate the expressions on type-in.
(Such as comma, or carriage return). Any arithmetic expression can
be input, as these examples show:

```
as $n
xa(0)=1.000
ya(0)=2.000
zb(0)=3.000

as
xa,xb,xc
xa+y=2
z++=1

# $n
xa(0)=1.000
ya(0)=2.000
zb(0)=3.000
xc(0)=6.000
dc(0)=10.000

as
xa,xb,xc
xa+y=1
yb=1
zc=1.0-1.0=c-1

# $n
xa(0)=2.000
ya(0)=3.000
zb(0)=4.000
xc(0)=7.000
dc(0)=11.000
```

SOME EXAMPLES:

ALL

C FOCAL=65 (V3D) 18-JUL-77

1.1 A "NOW PLEASE ENTER A NUMBER: ",X
1.2 T 1,"THE VALUE OF THE NUMBER SquARED IS",X*2,1!

C FOCAL=65 (V3D) 18-JUL-77

1.1 A "NOW PLEASE ENTER A NUMBER: ",X
1.2 T 1,"THE VALUE OF THE NUMBER SquARED IS",X*2,1!

GO
NOW PLEASE ENTER A NUMBER: 4
THE VALUE OF THE NUMBER SquARED IS 16.000

GO
NOW PLEASE ENTER A NUMBER: 4.5
THE VALUE OF THE NUMBER SquARED IS 20.250

GO
NOW PLEASE ENTER A NUMBER: -3
THE VALUE OF THE NUMBER SquARED IS 9.000

C FOCAL=65 (V3D) 18-JUL-77

1.1 A "NOW PLEASE ENTER A NUMBER: ",X
1.2 T 1,"THE VALUE OF THE NUMBER SquARED IS",X*2,1!

GO
NOW PLEASE ENTER A NUMBER: 4
THE VALUE OF THE NUMBER SquARED IS 16.000

C FOCAL=65 (V3D) 18-JUL-77

1.1 A "NOW PLEASE ENTER A NUMBER: ",X
1.2 T 1,"THE VALUE OF THE NUMBER SquARED IS",X*2,1!

GO
NOW PLEASE ENTER A NUMBER: -3
THE VALUE OF THE NUMBER SquARED IS 9.000
C FOCAL=65 (V3D) 18-JUL-77

1.10 A 1:"NOW PLEASE ENTER A NUMBER" ":X:"!
1.20 T 1:"THE VALUE OF THE NUMBER SQUARED IS":X*2,1:

*GO

NOW PLEASE ENTER A NUMBER: 2.2

THE VALUE OF THE NUMBER SQUARED IS= 4.840

*GO

NOW PLEASE ENTER A NUMBER: X*1

THE VALUE OF THE NUMBER SQUARED IS= 16.240

*T S

X*1 X= 5.280

*
A REASONABLE QUESTION AT THIS POINT MIGHT BE "HOW DOES
FOCAL INFORM ME WHEN IT ENCOUNTERS AN ERROR OF SOME KIND?", ERROR
MESSAGES IN FOCAL ARE ALWAYS STARTED WITH A '9' CHARACTER IN THE FIRST
POSITION ON THE LINE, THE '9' IS THEN FOLLOWED BY A CODE NUMBER, WHICH
INDICATES WHAT THE ERROR IS (A LIST OF ALL THE CODE NUMBERS AND THEIR
Meanings IS GIVEN IN AN APPENDIX). IF THE ERROR OCCURRED IN A STORED
LINE, THEN THE LINE NUMBER OF THE LINE IS ALSO OUTPUT, THEN THE LINE IS
OUTPUT TO THE USER'S CONSOLE, WITH AN 'U' (UPARROW) POINTING TO THE
POSITION IN THE LINE WHERE FOCAL WAS PROCESSING AT THE TIME THE ERROR
CONDITION WAS ENCOUNTERED. THIS INFORMATION IS USUALLY ENOUGH TO
QUICKLY DETERMINE THE CAUSE OF THE ERROR. SOME ILLUSTRATIONS:

C FOCAL=45 (V3D) 18-JUL-77
1.18 $ "PLEASE ENTER A NUMBER! ", " ,I
1.20 $ "THE VALUE OF THE NUMBER SQUARED IS
" , " ,I
$ 1.2
1.20 $ "THE VALUE OF THE NUMBER SQUARED IS
" , " ,I:
$ 1.3
$999.1 $ "HI" ,!
$999.1 $ "HI" ,!

(ErROr CODE 22 IS 'WRITE OF NON-EXISTENT LINE')
$ 2
$ (NO COMPLAINT HERE, THERE WAS NOTHING TO ERASE)
$ 999.1 $ "HI" ,!
$999.1 $ "HI" ,!

(ErROr CODE 4 IS 'ILLEGAL LINE NUMBER'). GROUP NUMBERS CAN ONLY
BE WITHIN THE RANGE 01-99,
$ 3
$HELLO THERE
$3
HELLO THERE

(ErROr CODE 3 IS 'UNRECOGNIZABLE COMMAND')
*C  WE HAVE BEEN SHOWING HOW TO TRANSFER CONTROL IN THE FOCAL PROGRAM TO A
*C  LINE OTHER THAN THE ONE THAT IS CURRENTLY BEING EXECUTED, VIA THE
*C  'GOTO' COMMAND. A VERY IMPORTANT FEATURE OF FOCAL IS THE
*C  ABILITY TO EVALUATE AN EXPRESSION AND TRANSFER CONTROL
*C  TO ONE OF SEVERAL PLACES, DEPENDING UPON THE RESULT OF THE EVALUATION.
*C  WHEN AN ARITHMETIC EXPRESSION IS EVALUATED IN FOCAL, IT IS REDUCED TO
*C  A SINGLE NUMERIC VALUE. THE 'IF' COMMAND (ABREVIATED '!') IN FOCAL
*C  ALLOWS THE TRANSFER OF CONTROL TO UP TO THREE DIFFERENT PLACES.
*C  DEPENDING ON WHETHER THE RESULT OF THE EXPRESSION IS LESS THAN ZERO,
*C  EQUAL TO ZERO, OR GREATER THAN ZERO, THE EXPRESSION IS ENCLOSED
*C  IN PARENTHESES, AND THE LINE NUMBERS OF THE PLACES TO GO FOLLOW THE
*C  CLOSING '!'. FOCAL WILL TRANSFER CONTROL TO THE FIRST LINE NUMBER IF THE
*C  RESULT OF THE EXPRESSION IS LESS THAN ZERO, TO THE SECOND LINE
*C  NUMBER IF THE RESULT OF THE EXPRESSION IS EQUAL TO ZERO, AND TO THE
*C  THIRD LINE NUMBER IF THE RESULT OF THE EXPRESSION IS GREATER THAN
*C  ZERO. THE LINE NUMBERS ARE SEPARATED BY COMMAS. SOME EXAMPLES:

*E  ALL

*1.1  A !,"ENTER A NUMBER!",X!
*1.2  IF (X)1.3,1.4,1.5
*1.3  T "THE NUMBER IS LESS THAN ZERO";!!!QUIT
*1.4  T "THE NUMBER IS EQUAL TO ZERO";!!!QUIT
*1.5  T "THE NUMBER IS GREATER THAN ZERO";!!!QUIT

*W

C FOCAL=65 (VSD) 18-JUL-77

1.10  A !,"ENTER A NUMBER!",X!
1.20  IF (X)1.3,1.4,1.5
1.30  T "THE NUMBER IS LESS THAN ZERO";!!!QUIT
1.40  T "THE NUMBER IS EQUAL TO ZERO";!!!QUIT
1.50  T "THE NUMBER IS GREATER THAN ZERO";!!!QUIT
*  
* GO 
ENTER A NUMBER:5 
THE NUMBER IS GREATER THAN ZERO  
* GO 
ENTER A NUMBER:5 
THE NUMBER IS LESS THAN ZERO  
* GO 
ENTER A NUMBER:0 
THE NUMBER IS EQUAL TO ZERO  
* GO 
ENTER A NUMBER:1+1 
THE NUMBER IS GREATER THAN ZERO  
* GO 
ENTER A NUMBER:5-5 
THE NUMBER IS EQUAL TO ZERO  
* W 
C FOCAL-65 (V3D) 18-JUL-77 
1.10 A 1,"ENTER A NUMBER:",X,1 
1.20 IF (X)1.31.4,1.5 
1.30 T "THE NUMBER IS LESS THAN ZERO";111QUIT 
1.40 T "THE NUMBER IS EQUAL TO ZERO";111QUIT 
1.50 T "THE NUMBER IS GREATER THAN ZERO";111QUIT 
*
CONTROL IS TRANSFERRED TO LINE '1.3' IF THE VALUE OF 'X' IS
LESS THAN ZERO, TO '1.4' IF IT IS EQUAL TO ZERO, AND TO '1.5' IF
IT IS GREATER THAN ZERO. AT THOSE LINES, THE 'TYPE' COMMAND OUTPUTS
AN APPROPRIATE MESSAGE. THEN THE 'QUIT' COMMAND IS USED TO
STOP THE EXECUTION ENTIRELY. IF THE 'QUIT' COMMAND WERE NOT THERE,
FOCAL WOULD HAVE CONTINUED EXECUTION WITH THE NEXT LINE IN
SEQUENCE. THE 'QUIT' COMMAND MAY BE USED AT ANY PLACE IN A FOCAL
PROGRAM TO CAUSE THE EXECUTION OF STORED INSTRUCTIONS TO STOP
ENTIRELY, AND FOCAL TO PROMPT WITH A '>', AND Await A NEW COMMAND.
MORE EXAMPLES:

1.20 IF X=71,3,1,4,1,9
M 1.3
1.30 T "THE NUMBER IS LESS THAN ZERO\\SEVEN",11QUIT
M 1.4
1.40 T "THE NUMBER IS EQUAL TO ZERO\\SEVEN",11QUIT
M 1.5
1.50 T "THE NUMBER IS GREATER THAN ZERO\\SEVEN",11QUIT
C FOCAL65 (v3d) 18-JUL-77

1.10 READ 1, "ENTER A NUMBER", X

1.20 IF (X=7), 1, 1.3, 1.9

1.30 T "THE NUMBER IS LESS THAN SEVEN", 111:QUIT

1.40 T "THE NUMBER IS EQUAL TO SEVEN", 111:QUIT

1.90 T "THE NUMBER IS GREATER THAN SEVEN", 111:QUIT

*GO

ENTER A NUMBER: 16

THE NUMBER IS LESS THAN SEVEN

*GO

ENTER A NUMBER: 19

THE NUMBER IS GREATER THAN SEVEN

*GO

ENTER A NUMBER: 17

THE NUMBER IS EQUAL TO SEVEN

*GO

ENTER A NUMBER: 15

THE NUMBER IS GREATER THAN SEVEN

*GO

ENTER A NUMBER: 19-12

THE NUMBER IS LESS THAN SEVEN
IT IS NOT ALWAYS NECESSARY TO SUPPLY ALL THREE LINE NUMBERS WHEN
USING THE 'IF' COMMAND. IF A LINE NUMBER IS OMITTED OR NULL (A COMMA
IS THERE, BUT NOTHING IS BEFORE IT), THEN FOCAL WILL PROCEED TO THE
NEXT COMMAND IN SEQUENCE (INSTEAD OF TRANSFERRING CONTROL TO
A NEW PLACE) IF THE CONDITION IS TRUE (EXPRESSION LESS THAN
ZERO, EQUAL TO ZERO, OR GREATER THAN ZERO). THIS ALLOWS FOR MORE
COMPACTNESS IN THE PROGRAM. SOME EXAMPLES SHOULD HELP CLARIFY THIS:

E ALL

1.1 A "ENTER A NUMBER!":X,1;I (Y-7)1,5,1.6;I "I HAVE CONTINUED ON 1.1":110
1.5 T "I AM AT LINE 1.5":110
1.6 T "I AM AT LINE 1.6":110

FOCAL-65 (VSD) 18-JUL-7:

1.1A "ENTER A NUMBER!":X,1;I (Y-7)1,5,1.6;I "I HAVE CONTINUED ON 1.1":110
1.5 T "I AM AT LINE 1.5":110
1.6 T "I AM AT LINE 1.6":110

GO

ENTER A NUMBER:13

I AM AT LINE 1.5

GO

ENTER A NUMBER:17

I AM AT LINE 1.6

GO

ENTER A NUMBER:19

I HAVE Continued ON 1.1
*C     SINCE THE THIRD LINE NUMBER WAS OMITTED, FOCAL CONTINUED WITH THE NEXT
*C     COMMAND IN SEQUENCE (THE "1" I MADE CONTINUED ON 1.1".) WHEN THE VALUE
*C     OF THE EXPRESSION (X-7) WAS GREATER THAN ZERO (I.E. X WAS GREATER THAN 7).
*C     MORE EXAMPLES:

*H 1.1

1.10 A "ENTER A NUMBER!";X;111 (X-7)1.5;1.6
*H 1.2 T "I AM AT LINE 1.2":110
*W

C FOCAL=65 (V30) 18 JUL 77

1.10 A "ENTER A NUMBER!";X;111 (X-7)1.5;1.6
1.20 T "I AM AT LINE 1.2":110
1.50 T "I AM AT LINE 1.5":110
1.60 T "I AM AT LINE 1.6":110

*G0

ENTER A NUMBER;

1 AM AT LINE 1.5
*G0

ENTER A NUMBER;

1 AM AT LINE 1.6
*G0

ENTER A NUMBER;

1 AM AT LINE 1.2

*C     IN THIS CASE THE NEXT COMMAND IN SEQUENCE JUST HAPPENED TO BE ON
*C     THE NEXT LINE, BUT THAT IS NO DIFFERENT THAN THE FIRST CASE WHERE THE
*C     NEXT COMMAND IN SEQUENCE IS ON THE SAME LINE. MORE EXAMPLES:

*H 1.1

1.10 A "ENTER A NUMBER!";X;111 (X-7)1.5
* *
*C FOCAL=65 (V30) 18-JUL-77
1.10 A "ENTER A NUMBER!";X;III (X-7)1.5
1.20 T "I AM AT LINE 1.2";110
1.50 T "I AM AT LINE 1.5";110
1.60 T "I AM AT LINE 1.6";110
*GO
ENTER A NUMBER
*GO
I AM AT LINE 1.5
*GO
ENTER A NUMBER
*GO
I AM AT LINE 1.2
*GO
ENTER A NUMBER
I AM AT LINE 1.2
*GO
1.10 A "ENTER A NUMBER!";X;III (X-7)1.5\1.6\"I HAVE CONTINUED ON 1.1",* GO
*C FOCAL=65 (V30) 18-JUL-77
1.10 A "ENTER A NUMBER!";X;III (X-7)1.5\1.6\"I HAVE CONTINUED ON 1.1",110
1.20 T "I AM AT LINE 1.2";110
1.50 T "I AM AT LINE 1.5";110
1.60 T "I AM AT LINE 1.6";110
*GO
ENTER A NUMBER
I HAVE CONTINUED ON 1.1
*GO
ENTER A NUMBER
I AM AT LINE 1.6
*GO
ENTER A NUMBER
I HAVE CONTINUED ON 1.1
* C SINCE THERE WAS A COMMA, BUT NO LINE NUMBER, THEN FOCAL PROCEEDS TO
* THE NEXT COMMAND IN SEQUENCE IF THE VALUE OF THE EXPRESSION IS LESS THAN
* ZERO. IF IT IS EQUAL TO ZERO, IT TRANSFERS TO LINE 1.6. IF IT IS GREATER
* THAN ZERO, FOCAL PROCEEDS TO THE NEXT COMMAND IN SEQUENCE. NOTE THIS:
* *1.1

1.10 A "ENTER A NUMBER!":X,111 (X=7) \1.6;I "I HAVE CONTINUED ON 1.1":110

C FOCAL-65 (V3D) 18-JUL-77
1.10 A " ENTER A NUMBER!":X,111 (X=7) \1.6;I " I HAVE CONTINUED ON 1.1":110
1.20 I " I AM AT LINE 1.2":110
1.50 I " I AM AT LINE 1.5":110
1.60 I " I AM AT LINE 1.6":110

*ON

ENTER A NUMBER:3
I AM AT LINE 1.6
*G

ENTER A NUMBER:7
I HAVE CONTINUED ON 1.1
*G

ENTER A NUMBER:9
I HAVE CONTINUED ON 1.1
*
THAT LITTLE COMMA WAS VERY IMPORTANT: IN THIS CASE FOCAL TRANSFERS
CONTROL TO LINE 1.6 IF THE VALUE OF THE EXPRESSION IS LESS THAN ZERO,
BUT PROCEEDS TO THE NEXT COMMAND IN SEQUENCE IF THE VALUE IS EQUAL TO,
OR GREATER THAN, ZERO (SAME AS ANOTHER EXAMPLE ABOVE). ANOTHER EXAMPLE:

1.10 A "ENTER A NUMBER!";X,111 (X-7)1.6,1.6:1T "I HAVE CONTINUED ON 1.1",!

C FOCAL=55 (V3D) 18-JUL-77

1.13 A " ENTER A NUMBER!";X,111 (X-7)1.6,1.6:1T "I HAVE CONTINUED ON 1.1",!
1.20 T " I AM AT LINE 1.2";11Q
1.30 T " I AM AT LINE 1.3";11Q
1.60 T " I AM AT LINE 1.6";11Q

GO

ENTER A NUMBER:3
I AM AT LINE 1.6

GO

ENTER A NUMBER:7
I HAVE CONTINUED ON 1.1

GO

ENTER A NUMBER:9
I AM AT LINE 1.6
FOCAL WILL TRANSFER CONTROL TO LINE 1.6 IF THE VALUE OF THE
EXPRESSION IS NOT EQUAL TO ZERO (I.E., LESS THAN OR GREATER THAN), BUT WILL PROCEED WITH THE NEXT COMMAND IN SEQUENCE IF THE VALUE IS EQUAL TO ZERO. THE 'IF' STATEMENT ALLOWS EITHER TWO OR THREE WAY BRANCHING OF PROGRAM CONTROL DEPENDING UPON THE VALUE OF AN ARITHMETIC EXPRESSION. THIS ALLOWS THE COMPUTER PROGRAM TO COMPARE QUANTITIES AND PERFORM DIFFERENT COMMAND SEQUENCES, DEPENDING UPON THE RELATIONSHIP OF THOSE QUANTITIES. Thus, THE 'IF' COMMAND IS A CONDITIONAL 'GOTO' COMMAND.

IT IS DESIRABLE TO HAVE FOCAL REMEMBER WHERE IT IS EXECUTING COMMANDS ON A GIVEN LINE. TRANSFER CONTROL TO, PERHAPS, ANOTHER LINE OR GROUP, THEN HAVE FOCAL RETURN TO THE PLACE IT REMEMBERED IT WAS AT, CONTINUING TO EXECUTE COMMANDS AS BEFORE. THIS IS A VERY POWERFUL FEATURE, SINCE IT ALLOWS THE PROGRAMMER TO WRITE A LINE (OR GROUP OF LINES) TO DO A SPECIFIC TASK, AND WHENEVER THAT TASK NEEDS TO BE DONE, PERFORM THE LINE OR GROUP, AND RETURN TO THE NEXT COMMAND IN SEQUENCE. THIS CAPABILITY IS PROVIDED FOR BY THE 'DO' COMMAND IN FOCAL. THE 'DO' COMMAND CAN BE USED TO PERFORM A SINGLE LINE, OR AN ENTIRE GROUP OF LINES, WHEN FOCAL ENCOUNTERS A 'DO' COMMAND, IT LOOKS FOR A LINE NUMBER OR A GROUP NUMBER (SUCH AS 13) FOLLOWING THE 'DO' COMMAND. FOCAL THEN REMEMBERS THE POSITION TO RETURN TO AFTER PERFORMING THE LINE OR GROUP, TRANSFERS CONTROL TO THE BEGINNING OF THE LINE OR GROUP, PERFORMS THE FOCAL COMMANDS THERE, THEN RETURNS TO THE NEXT COMMAND IN SEQUENCE FOLLOWING THE 'DO' COMMAND.

THE 'DO' COMMAND OPERATES SLIGHTLY DIFFERENTLY WHEN PERFORMING ONLY A SINGLE LINE OF COMMANDS. THAN IT DOES WHEN PERFORMING AN ENTIRE GROUP OF COMMANDS. LET US LOOK AT HOW THE 'DO' COMMAND FUNCTIONS WHEN WE SPECIFY A SPECIFIC LINE NUMBER. THE 'DO' CAUSES FOCAL TO REMEMBER WHERE TO RETURN TO AFTER THE 'DO' HAS COMPLETED, TRANSFERS CONTROL TO THE FIRST COMMAND BEYOND THE SPECIFIED LINE, THEN PROCEEDS TO EXECUTE COMMANDS AS THEY ARE ENCOUNTERED UNTIL A CARTRIDGE RETURN IS ENCOUNTERED. AT THE TIME THE CARTRIDGE RETURN IS ENCOUNTERED, CONTROL RETURNS TO THE PLACE FOCAL REMEMBERED WHEN THE 'DO' WAS ENCOUNTERED. SOME EXAMPLES!

ALL

1.1 * "I AM EXECUTING COMMANDS ON LINE 2.1",
1.2 * "I AM EXECUTING COMMANDS ON LINE 2.2",
1.3 * "I AM EXECUTING COMMANDS ON LINE 2.3",

2.1 * "I AM EXECUTING COMMANDS ON LINE 2.1",
2.2 * "I AM EXECUTING COMMANDS ON LINE 2.2",
2.3 * "I AM EXECUTING COMMANDS ON LINE 2.3",

NOTE THE ACTION OF THE 'GOTO' COMMAND:

GOTO

1 2.1 EXECUTING COMMANDS ON LINE 2.1
1 2.2 EXECUTING COMMANDS ON LINE 2.2
1 2.3 EXECUTING COMMANDS ON LINE 2.3

GOTO 2.2
1 2.1 EXECUTING COMMANDS ON LINE 2.1
1 2.2 EXECUTING COMMANDS ON LINE 2.2
1 2.3 EXECUTING COMMANDS ON LINE 2.3

GOTO 2.3
1 2.1 EXECUTING COMMANDS ON LINE 2.1
1 2.2 EXECUTING COMMANDS ON LINE 2.2
1 2.3 EXECUTING COMMANDS ON LINE 2.3
THE GOTO COMMAND DOES NOT REMEMBER ANY PLACE TO RETURN TO.

NOW NOTE THE ACTION OF THE 'DO' COMMAND:

DO 2.1
1 AM EXECUTING COMMANDS ON LINE 2.1

DO 2.2
1 AM EXECUTING COMMANDS ON LINE 2.2

DO 2.3
1 AM EXECUTING COMMANDS ON LINE 2.3

DO 2.2 "THE 'DO' COMMAND RETURNED HERE"; DO 2.3 "MORE COMMANDS ON THIS LINE"

1 AM EXECUTING COMMANDS ON LINE 2.2
THE 'DO' COMMAND RETURNED HERE
1 AM EXECUTING COMMANDS ON LINE 2.3
MORE COMMANDS ON THIS LINE

2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1"; 110 2.3

OFF-CAL-69 (V30) 19-JUL-79

2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1"; 110 2.3
2.20 T "I AM EXECUTING COMMANDS ON LINE 2.2"; 1
2.30 T "I AM EXECUTING COMMANDS ON LINE 2.3"; 1

DO 2.1
1 AM EXECUTING COMMANDS ON LINE 2.1
1 AM EXECUTING COMMANDS ON LINE 2.3
THE 'DO 2.1' REMEMBERED WHERE TO COME BACK TO, TRANSFERRED CONTROL
TO LINE 2.1, AND BEGAN EXECUTING THE COMMANDS THERE. THE 'GOTO' COMMAND
AT THE END OF LINE 2.1 TRANSFERRED CONTROL TO LINE 2.3 WITHOUT FOCAL EVER
HAVING ENCOUNTERED A CARRIAGE RETURN. THIS COMMANDS ON LINE 2.2 WHERE
EXECUTED UNTIL A CARRIAGE RETURN WAS ENCOUNTERED AT THE END OF LINE 2.3,
AT WHICH TIME FOCAL RETURNED TO THE PLACE IT REMEMBERED TO GO BACK TO
(AFTER THE 'DO 2.1'). SAW NO MORE COMMANDS THERE, SO IT QUIT.
MORE EXAMPLES:

C FOCAL=65 (Y3D) 18-JUL-77

2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1", I1G 2.3
2.20 T "I AM EXECUTING COMMANDS ON LINE 2.2", 1
2.30 T "I AM EXECUTING COMMANDS ON LINE 2.3", 1

G 2.1
I AM EXECUTING COMMANDS ON LINE 2.1
I AM EXECUTING COMMANDS ON LINE 2.3

G 2.2
I AM EXECUTING COMMANDS ON LINE 2.2
I AM EXECUTING COMMANDS ON LINE 2.3

DO 2.2
I AM EXECUTING COMMANDS ON LINE 2.2

DO 2.3 DO 2.2 I1G "THAT'S ALL!"
I AM EXECUTING COMMANDS ON LINE 2.3
I AM EXECUTING COMMANDS ON LINE 2.2
THAT'S ALL!
*E ALL
*2.1 T 1: "THE VALUE OF X =", X, 1

C FOCAL=65 (V3D) 18-JUL-77
2.10 T 1: "THE VALUE OF X =", X, 1
*8 X=100 2.11 S X=100 2.1
THE VALUE OF X = 1.000
THE VALUE OF X = 1.000
*11 S X=X+1.0 2.110 1.1

C FOCAL=65 (V3D) 18-JUL-77
1.10 S X=X+1.0 2.110 1.1
2.10 T 1: "THE VALUE OF X =", X, 1

*GO
THE VALUE OF X = 1.000
THE VALUE OF X = 2.000
THE VALUE OF X = 3.000
THE VALUE OF X = 4.000
THE VALUE OF X = 5.000
THE VALUE OF X = 6.000
THE VALUE OF X = 7.000
THE VALUE OF X = 8.000
THE VALUE OF X = 9.000
THE VALUE OF X = 10.000
THE VALUE OF X = 11.000
THE VALUE OF X = 12.000

1-79 * 2.10 /
T 1: "THE VALUE OF X =", X, 1
THE SEQUENCE OF COMMANDS IN LINE '1.1' ADDED ONE TO THE VALUE OF 'X',
REPEATED THE COMMANDS ON LINE '2.1', THEN TRANSFERRED BACK TO THE BEGINNING
OF LINE '1.1' AGAIN. THIS SEQUENCE OF COMMANDS WOULD HAVE
CONTINUED TO DO THIS AD INFINITUM. THIS IS CALLED AN 'INFINITE LOOP',
SINCE THERE IS NO WAY (NORMALLY) TO STOP EXECUTION. IN THIS CASE,
I PRESSED THE 'INTERRUPT' BUTTON ON MY COMPUTER, WHICH CAUSED A
FATAL ERROR, STOPPING THE EXECUTION, AND PRINTING THE APPROPRIATE
ERROR MESSAGES. WHAT IF HE WANTED TO OUTPUT ONLY THE FIRST 10 VALUES
OF 'X', THEN CAUSE THE PROGRAM TO STOP? EXAMPLE:

1.10 S X=X+1:D 2.11I (X=10)1.11T "THAT'S ALL!":1!1Q

1.12 S X=X+1:D 2.12I (X=10)1.12T "THAT'S ALL!":1!1Q

2.12 T 1,"THE VALUE OF X= ";X;1

THE VALUE OF X= 1.000
THE VALUE OF X= 2.000
THE VALUE OF X= 3.000
THE VALUE OF X= 4.000
THE VALUE OF X= 5.000
THE VALUE OF X= 6.000
THE VALUE OF X= 7.000
THE VALUE OF X= 8.000
THE VALUE OF X= 9.000
THE VALUE OF X= 10.000
THAT'S ALL!
THE 'IF' STATEMENT WAS USED TO COMPARE THE VALUE OF 'X' TO THE
NUMBER 10, AND AS LONG AS 'X' WAS LESS THAN 10 (X<10) LESS
THAN ZERO, THEN THE 'IF' COMMAND WOULD TRANSFER CONTROL BACK TO
THE BEGINNING OF LINE 1:1, WHEN THE VALUE OF 'X' WAS EQUAL TO 10
(X=10) EQUAL TO ZERO), THEN THE NEXT COMMAND IN SEQUENCE WAS EXECUTED.
WHICH TYPED 'THAT'S ALL!', AND QUIT.
BACK TO THE 'DO' COMMAND. IT DOES NOT MATTER WHEN OR WHERE THE
CARRIAGE RETURN IS ENCOUNTERED. ONCE CONTROL HAS BEEN TRANSFERRED TO THE
SPECIFIC LINE TO 'DO', BUT WHEN IT DOES ENCOUNTER A CARRIAGE
RETURN, THEN CONTROL RETURNS TO THE NEXT COMMAND IN SEQUENCE AFTER THE
'DO' WHICH TRANSFERRED CONTROL. WHEN THE 'DO' COMMAND IS FOLLOWED BY
A GROUP NUMBER (SUCH AS '4'), THEN FOCAL REMEMBERS WHERE TO COME BACK TO
AND TRANSERS CONTROL TO THE LOWEST NUMBERED STEP WITHIN THAT GROUP. COMMAND
ARE EXECUTED AS THEY ARE ENCOUNTERED, UNTIL A CARRIAGE RETURN IS ENCOUNTERED
IN A LINE WHICH IS NOT PART OF THAT GROUP (I.E., IT WAS A DIFFERENT GROUP NUMBER
AT THAT POINT, CONTROL IS RETURNED TO THE PLACE REMEMBERED). AT ANY TIME,
WHILE A 'DO' IS BEING PERFORMED, IMMEDIATE RETURN TO THE PLACE
REMEMBERED CAN BE FORCED BY USING THE 'RETURN' COMMAND. SOME EXAMPLES:

2.2 T 'THE VALUE OF X+1 IS = "X*.X+,!

C FOCAL=65 (V3D) 18=JUL=77
1.10 S X*X+1;D 2.1;1 (X+1)😀 "THAT'S ALL!";I;Q
2.10 T 'THE VALUE OF X+1 IS = "X+.X;1
2.20 T 'THE VALUE OF X+2 IS = "X*.X+,!

M 1.1
1.10 S X*X+1;D 2.1;1 (X+1)😀 "THAT'S ALL!";I;Q

C FOCAL=65 (V3D) 18=JUL=77
1.10 S X*X+1;D 2.1;1 (X+1)😀 "THAT'S ALL!";I;Q
2.10 T 'THE VALUE OF X+1 IS = "X+.X;1
2.20 T 'THE VALUE OF X+2 IS = "X*.X+,!


The value of x = 1,000
The value of x+2 is = 1,000
The value of x = 2,000
The value of x+2 is = 4,000
The value of x = 3,000
The value of x+2 is = 9,000
The value of x = 4,000
The value of x+2 is = 16,000
The value of x = 5,000
The value of x+2 is = 25,000
The value of x = 6,000
The value of x+2 is = 36,000
The value of x = 7,000
The value of x+2 is = 49,000
The value of x = 8,000
The value of x+2 is = 64,000
The value of x = 9,000
The value of x+2 is = 81,000
The value of x = 10,000
The value of x+2 is = 100,000
That's all!
1.10 S X=X+1 10 2:1 I (X-184\4)1:1 IT "THAT'S ALL!" 1:1 10
2.15 I (X-312:2)1R

C FOCAL69 (V3D) 18-JUL=77
1.10 S X=X+1 10 2:1 I (X-511:111 IT "THAT'S ALL!" 1:1 10
2.18 T I:"THE VALUE OF X= " ,X,1
2.15 I (X-312:2)1R
2.20 T I:"THE VALUE OF X+2 IS = " ,X+2,1

G

THE VALUE OF X=    1.000
THE VALUE OF X+2 IS =    1.000
THE VALUE OF X=    2.000
THE VALUE OF X+2 IS =    4.000
THE VALUE OF X=    3.000
THE VALUE OF X=    4.000
THE VALUE OF X=    5.000
THAT'S ALL

C

IN GROUP 2, AT LINE 2:15, THE 'IF' STATEMENT TRANSFERRED CONTROL
TO LINE 2:2 (OUTPUTTING X+2) AS LONG AS X WAS LESS THAN 3. IN
ALL OTHER CASES, AND IMMEDIATE 'RETURN' FROM GROUP 2 WAS EXECUTED.

G

8 X=2810 2

THE VALUE OF X=    20.000

S X=110 2

THE VALUE OF X=    1.000

THE VALUE OF X+2 IS =    1.000

S X=2810 2.2

THE VALUE OF X+2 IS =    400.000
"ANOTHER 'DO' COMMAND CAN BE ENCOUNTERED ANYTIME AFTER A 'DO'.
"A COMMAND HAS BEEN TRANSFERRED CONTROL TO A LINE OR GROUP. FOCAL PROCESSES
"A 'DO' COMMAND, BY REMEMBERING THERE TO COME BACK TO, TRANSFERING CONTROL
"A TO THE LINE OR GROUP, AND WHEN CONTROL RETURNS, IT WILL BE TO THE LAST
"A LINE REMEMBERED. THEN WHEN THE FIRST 'DO' IS OVER, CONTROL WILL RETURN
"A TO THE NEXT COMMAND IN SEQUENCE AFTER THE FIRST 'DO'. THUS 'DO'
"A COMMANDS CAN BE NESTED. THERE IS NO IMPLIRED LIMIT ON THE DEPTH TO
"A WHICH FOCAL 'DO' COMMANDS MAY BE NESTED. SOME EXAMPLES:
"

C FOCAL=65 (V3D) 18 JUL 77
1.10 S X=X+1;D 2;I (X=511;11) "THAT'S ALL!":110
2.10 T I "THE VALUE OF X= ":X;1
2.15 I (X-3)2;2;R
2.20 T I "THE VALUE OF X+2 IS ":X+2;1
3.10 T I "THE VALUE OF X+3 IS ":X+3;1
4 Go
THE VALUE OF X= 1.000
THE VALUE OF X+2 IS = 1.000
THE VALUE OF X= 2.000
THE VALUE OF X+2 IS = 4.000
THE VALUE OF X= 3.000
THE VALUE OF X+3 IS = 27.000
THE VALUE OF X= 4.000
THE VALUE OF X+3 IS = 64.000
THE VALUE OF X= 5.000
THE VALUE OF X+3 IS = 125.000
THAT'S ALL!
"
* Now, if the value of $X$ is greater than, or equal to 3, the value
* of $X+3$ is output instead of the value of $X+2$. The 'DO' command
* is a very useful facility, and greatly increases the power
* of FOCAL. The 'IF' command provides a facility to perform a 'GOTO'
* based upon the value of an arithmetic expression. The 'ON' command
* provides the ability to perform a 'DO' command based on the value
* of an arithmetic expression. It works in the same manner as the 'IF'
* command, but instead of transferring completely to the specified line
* or group, a 'DO' command of the specified line or group is performed.
* And, when the 'DO' comes back, the next statement in sequence is
* executed by FOCAL, as in normal sequential processing. Some examples:
*
* ALL
*
C FOCAL=65 (V30) 18-JUL-77

1.1 A !"NUMBER1",X,110 (X=7)2;1,2,2,316 1.1

2.1 T "I AM AT LINE 2.1"$!1
2.2 T "I AM AT LINE 2.2"$!1
2.3 T "I AM AT LINE 2.3"$!1
*

C FOCAL=65 (V30) 18-JUL-77

1.10 A !"NUMBER1",X,110 (X=7)2;1,2,2,316 1.1

2.10 T "I AM AT LINE 2.1"$!1
2.20 T "I AM AT LINE 2.2"$!1
2.30 T "I AM AT LINE 2.3"$!1
*

NUMBER=3
I AM AT LINE 2.1

NUMBER=7
I AM AT LINE 2.2

NUMBER=9
I AM AT LINE 2.3

NUMBER=4
I AM AT LINE 2.1

NUMBER:
$!-19$ # 1.10
A !"NUMBER1",X,110 (X=7)2;1,2,2,316 1.1

THE 'INTERRUPT' BUTTON WAS PRESSED TO GET OUT OF THE ABOVE
INFINITE LOOP. A 'DO' OF LINE 2.1 WAS PERFORMED IF THE VALUE OF
X WAS LESS THAN 7, A 'DO' OF LINE 2.2 WAS PERFORMED IF THE
VALUE WAS EQUAL TO 7, AND A 'DO' OF LINE 2.3 WAS PERFORMED IF THE
VALUE WAS GREATER THAN 7. IN ALL CASES, CONTROL RETURNED TO THE
NEXT STATEMENT IN SEQUENCE WHICH FOLLOWED THE 'ON' COMMAND. MORE
EXAMPLES:

M 1.1

1.10 A "NUMBER", X, 1:0 (X=7) 2.1 2.2 2.3 1.1

FOCAL=65 (V30) 18-JUL-77

1.10 A "NUMBER", X, 1:0 (X=7) 2.1 2.2 2.3 1.1

2.10 T "I AM AT LINE 2.1"
2.20 T "I AM AT LINE 2.2"
2.30 T "I AM AT LINE 2.3"

NUMBER 13
1 AM AT LINE 2.1

NUMBER 17
1 AM AT LINE 2.1
1 AM AT LINE 2.2
1 AM AT LINE 2.3

NUMBER 19
1 AM AT LINE 2.3

NUMBER 7-10 1.10
A "NUMBER", X, 1:0 (X=7) 2.1 2.2 2.3 1.1
IN THIS CASE, ALL OF GROUP 2 WAS PERFORMED WHEN THE VALUE OF X
WAS EQUAL TO SEVEN. LINE AND/OR GROUP NUMBERS MAY BE OMITTED
JUST AS IN THE 'IF' COMMAND), IN WHICH CASE CONTROL WILL SIMPLY
PASS TO THE NEXT COMMAND IN SEQUENCE (JUST AS IN THE 'IF' COMMAND).

**SOME EXAMPLES:**

```
1.10 A "%NUMBER!",X:110 (X-7)2,1\,2,2,3\,10 1,1
```

```
2.10 T "I AM AT LINE 2.1",1
2.20 T "I AM AT LINE 2.2",1
2.30 T "I AM AT LINE 2.3",1
```

**GO**

```
NUMBER:9
I AM AT LINE 2.3
```

```
NUMBER:7
I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3
```

```
NUMBER:13
```

```
NUMBER:12
```

```
NUMBER:18
I AM AT LINE 2.3
```

```
NUMBER:17
I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3
```

```
NUMBER:19  1.10
A "%NUMBER!",X:110 (X-7)2,2,3\,10 1,1
```

```
CONTROL SIMPLY PASSES TO THE NEXT COMMAND IN SEQUENCE (THE 'GO TO')
WHEN THE VALUE OF X WAS LESS THAN SEVEN, THERE WAS NOTHING DONE
WHEN THE VALUE OF X WAS LESS THAN SEVEN, HERE IS A VERY POWERFUL
FEATURE OF FOCAL. ANYPLACE A LINE NUMBER OR A GROUP NUMBER COULD
NORMALLY BE USED IN A FOCAL STATEMENT, AN ARITHMETIC EXPRESSION CAN
BE USED THERE INSTEAD. THE ARITHMETIC EXPRESSION IS REDUCED TO A SINGLE
NUMBER OF THE FORM '.89,99' AND THAT VALUE IS USED AS THE LINE AND/OR
GROUP NUMBER, THIS ALLOWS SUCH PARAMETERS TO BE VARIABLE QUANTITIES.
NOTE: IF AN ARITHMETIC EXPRESSION IS USED, IT MUST NOT BEGIN WITH A
DIGIT, THUS 'X+.1' IS OK. BUT '1+X' IS NOT.

SOME EXAMPLES:

A FOCAL=69 (V30) 18-JUL-77

2.10 T *I AM AT LINE 2.1*
2.20 T *I AM AT LINE 2.2*
2.30 T *I AM AT LINE 2.3*

S X=2
T 5

X0(0)= 2.000

D0 X
I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3

D X=1
I AM AT LINE 2.1

D X=.2
I AM AT LINE 2.2

D X=3
I AM AT LINE 2.3

G 2.1
I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3

G X=1
I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3
THE 'Y' ABOVE SIMPLY SET THE VALUE OF 'Y' TO 2. INPUT THAT
VALUE IN THE 'ASK' COMMAND, ASSIGNED IT TO THE VARIABLE 'X', THE
VALUE OF THE EXPRESSION IN THE 'ON' COMMAND WAS THEN GREATER THAN
ZERO, SO FOCAL CONTINUED ON LINE 1.1. HOWEVER, THE VALUE OF Y WAS
BEEN CHANGED TO 2, SO WHEN THE NEXT TIME A NUMBER WAS ASKED FOR, A
ZERO WAS ENTERED, 'X' WAS SET TO ZERO, AND THE 'ON' COMMAND
PERFORMED GROUP 'Y' (GROUP 2 IN THIS CASE), AND THEN RETURNED TO
CONTINUE PROCESSING THE NEXT SEQUENTIAL STATEMENT.
THE CAPABILITY TO ALLOW AN EXPRESSION TO DETERMINE THE VALUE OF
A LINE OR GROUP NUMBER EXPANDS THE POWER OF FOCAL.

SOMETIMES A COMPUTER PROGRAM MUST REPEAT A PROCESS OVER AND OVER
REMEMBER, COMPUTERS ARE GOOD AT THIS!, WE HAVE SEEN
ONE WAY TO DO THIS IS FOCAL ALREADY. A VARIABLE CAN BE USED AS A
COUNTER AND GET INCREMENTED (OR DECREMENTED) EACH TIME THE PARTICULAR
PROCESS IS DONE. AN 'IF' OR 'ON' STATEMENT CAN BE USED TO DETERMINE
IF THE PROCESS IS TO BE DONE AGAIN BY TESTING THE VALUE OF THE
COUNTING VARIABLE. AN EXAMPLE FOLLOWS:

FOCAL=65 (V3D) 18-JUL-77
1.1 A !"NUMBER",C:!
1.1 T "ENTER BEGINNING, INCREMENT, AND ENDING VALUES! ",B,I,E:
1.12 T "A PROCESS WITH B=",B,II ((B=B+1)-E)1,2,1.2Q

FOCAL=65 (V3D) 18-JUL-77
1.10 A "ENTER BEGINNING, INCREMENT, AND ENDING VALUES! ",B,I,E:
1.20 T "A PROCESS WITH B=",B,II ((B=B+1)-E)1,2,1.2Q
ENTER BEGINNING, INCREMENT, AND ENDING VALUES: 1, 1.5

A PROCESS WITH B = 1.000
A PROCESS WITH B = 2.000
A PROCESS WITH B = 3.000
A PROCESS WITH B = 4.000
A PROCESS WITH B = 5.000

ENTER BEGINNING, INCREMENT, AND ENDING VALUES: 1, 2.7

A PROCESS WITH B = 1.000
A PROCESS WITH B = 3.000
A PROCESS WITH B = 5.000
A PROCESS WITH B = 7.000

ENTER BEGINNING, INCREMENT, AND ENDING VALUES: 1, 2.10

A PROCESS WITH B = 1.000
A PROCESS WITH B = 3.000
A PROCESS WITH B = 5.000
A PROCESS WITH B = 7.000
A PROCESS WITH B = 9.000
THE ABOVE EXAMPLE ASKED FOR THREE VALUES, A BEGINNING ('B'), AN INCREMENT ('I'), AND AN ENDING VALUE ('E'). THE COMMANDS AT LINE 1.2 FORM A LOOP, WHERE THE 'I' COMMAND IS EXECUTED, THE INCREMENT IS ADDED TO 'B' (AND BECOMES THE NEW VALUE OF 'B'), AND THEN 'B' IS COMPARED TO 'E'. IF 'B' IS LESS THAN, OR EQUAL TO, 'E', THEN CONTROL GETS TRANSFERRED BACK TO LINE 1.2 AND THE 'I' COMMAND IS EXECUTED AGAIN. IF THE VALUE OF 'B' IS GREATER THAN 'E', THEN CONTROL PROCEEDS TO THE NEXT STATEMENT IN SEQUENCE. WHICH STOPS THE PROGRAM. THIS TYPE OF LOOP IS OFTEN REQUIRED IN COMPUTER PROGRAMS, SO FOCAL PROVIDES A MORE COMPACT METHOD FOR DOING A LOOP OF THIS TYPE. THE 'FOR' COMMAND ALLOWS THE PROGRAMMER TO PERFORM A LOOP IN THIS MANNER. LET'S FIRST LOOK AT THE ABOVE EXAMPLE, BUT WITH A 'FOR' LOOP USED INSTEAD OF THE 'IF' LOOP.

```
A
1.1 A "BEGINNING, INCREMENT, ENDING VALUE: "I,EI FOR X=X',.EIT "A PROCESS WITH X=X',X'.I

C FOCAL-65 (V3D) 18-JUL-77
1.10 A "BEGINNING, INCREMENT, ENDING VALUE: "I,EI FOR X=X',.EIT "A PROCESS WITH X=X',X'.I

GO
BEGINNING, INCREMENT, ENDING VALUE: 1.1.9
A PROCESS WITH X= 1.000
A PROCESS WITH X= 2.000
A PROCESS WITH X= 3.000
A PROCESS WITH X= 4.000
A PROCESS WITH X= 5.000

GO
BEGINNING, INCREMENT, ENDING VALUE: 1.2.7
A PROCESS WITH X= 1.000
A PROCESS WITH X= 2.000
A PROCESS WITH X= 3.000
A PROCESS WITH X= 4.000
A PROCESS WITH X= 5.000

GO
BEGINNING, INCREMENT, ENDING VALUE: 3.1.3
A PROCESS WITH X= -3.000
A PROCESS WITH X= -2.000
A PROCESS WITH X= -1.000
A PROCESS WITH X= 0.000
A PROCESS WITH X= 1.000
A PROCESS WITH X= 2.000
A PROCESS WITH X= 3.000
```
THE 'FOR' COMMAND IS FOLLOWED BY A VARIABLE NAME WHICH IS USEC
AS THE COUNTING VARIABLE. UP TO THREE OPTIONS MAY BE SPECIFIED
FOLLOWING THE 'x', THERE ARE BEGINNING VALUE TO BE ASSIGNED TO
THE COUNTING VARIABLE, THE INCREMENT THAT IS TO BE ADDED ON TO THE
COUNTING VARIABLE EACH TIME CONTROL IS RETURNED, AND THE ENDING VALUE
WHICH DETERMINES WHEN THE LOOPING PROCESS WILL NORMALLY TERMINATE.
THE EXACT OPERATION OF THE 'FOR' COMMAND IS AS FOLLOWS. THE
COUNTING VARIABLE ('x' IN ABOVE EXAMPLE) IS SET EQUAL TO THE BEGINNING
VALUE ('b' IN EXAMPLE), THE INCREMENT AND THE ENDING VALUE ARE REMEMBERED
BY FOCAL, AS WELL AS THE START OF THE NEXT STATEMENT ON THE LINE.
A 'GO' OF ALL THE COMMANDS ON THE REST OF THE LINE IS PERFORMED. THIS
CONTROL RETURNS WHEN A CARRIAGE RETURN IS ENCOUNTERED, WHEN CONTROL
RETURNS, THE INCREMENT IS ADDED TO THE COUNTING VARIABLE (THE INCREMENT
MAY BE NEGATIVE, TO COUNT BACKWARDS), AND THEN THE COUNTING VARIABLE IS
COMPARSED TO THE ENDING VALUE, IF THE COUNTING VARIABLE IS LESS THAN.
OR EQUAL TO, THE ENDING VARIABLE (GREATER THAN OR EQUAL TO, IF THE
INCREMENT WAS NEGATIVE), THEN ANOTHER 'GO' OF THE REMAINDER OF THE LINE IS PER
OTHERWISE, THE 'FOR' LOOP HAS COME TO COMPLETION, AND CONTROL IS
TRANSFERRED TO THE BEGINNING OF THE NEXT FOCAL LINE NUMBER IN SEQUENCE.
NOTE THAT CONTROL IS TRANSFERRED TO THE NEXT LINE AND NOT TO THE NEXT
COMMAND, SINCE THE NEXT COMMAND, AND ALL THOSE ON THE REST OF THE LINE
WERE PART OF THE 'FOR' LOOP. IF THE INCREMENT IS OMITTED (ONLY 2
PARAMETERS SPECIFIED), THEN A VALUE OF 1 IS ASSUMED.

SOME EXAMPLES:

� A
� X=1,2,10:IT X:
    1.000
    3.000
    5.000
    7.000
    9.000

� X=1,10:IT X:
    1.000
    2.000
    3.000
    4.000
    5.000
    6.000
    7.000
    8.000
    9.000
   10.000

�
$\text{LET } X = 10, Y = 1, T = X, !$

10.000
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000

$\text{LET } Y = 7$

$\text{LET } X = 1, Y = T X, !$

1.000
2.000
3.000
4.000
5.000
6.000
7.000

$\text{LET } X = 10, Y = 10, T = X, !$

10.000
11.000
12.000
13.000
14.000
15.000
16.000
17.000

$\text{C} \text{ SINCE A 'DO'} \text{ OF THE REMAINDER OF THE LINE IS PERFORMED, AND SINCE}$

$\text{DO'S MAY BE NESTED, THEN THERE MAY BE A 'FOR' LOOP WITHIN A}$

$\text{'FOR'} \text{ LOOP. SOME EXAMPLES!}$

$\text{LET } X = 1, 5, T = "X\#!!, X, IF } Y = 1, 3, T = "Y\#!!, Y, !$

$X = 1.000$   $Y = 1.000$
$X = 2.000$   $Y = 2.000$
$X = 3.000$   $Y = 3.000$
$X = 4.000$   $Y = 4.000$
$X = 5.000$   $Y = 5.000$
$X = 1.000$   $Y = 2.000$
$X = 3.000$   $Y = 3.000$
$X = 5.000$   $Y = 5.000$
$X = 1.000$   $Y = 2.000$
$X = 3.000$   $Y = 3.000$
*C OR, WATCH THIS ONE!
*C X=-8.8*1T "="!1;F Y=8.X+2*1T "*

A ONE LINE FOCAL PROGRAM PLOTS A PARABOLA!
AS YOU HAVE SEEN, ANY OR ALL OF THE PARAMETERS IN THE 'FOR' COMMAND
CAN BE ARITHMETIC EXPRESSIONS (VERY POWERFUL). IT MAY SEEM, AT FIRST
GLANCE, THAT THE FACT THAT THE 'FOR' COMMAND CAN ONLY 'DO' THE COMMANDS
ON THE REMAINDER OF THE LINE IS A SERIOUS LIMITATION. NOT SO, BECAUSE
ONE OF THOSE COMMANDS CAN BE A 'DO' COMMAND TO PERFORM AS MANY
LINES OR GROUPS AS THE PROGRAMMER WANTS, AND CONTROL WILL RETURN
TO THE COMMAND FOLLOWING THE 'DO', EXAMPLE:

*E A

*2.1 T "AT LINE 2.1":!
*2.2 T "AT LINE 2.2":!
*2.3 T "AT LINE 2.3":!
*F X=1,3 DO 2
AT LINE 2.1
AT LINE 2.2
AT LINE 2.3
AT LINE 2.1
AT LINE 2.2
AT LINE 2.3
AT LINE 2.1
AT LINE 2.2
AT LINE 2.3
A 'FOR' loop may be terminated at any time if a 'RETURN' command is encountered. This returns control back from the 'DO' of the remainder of the commands after the 'FOR'. Thus:

1.1 F X=1,10; "X="X,"i! 1,(X-6),1.3
1.2 T "THE FOR LOOP CAME HERE WHEN COMPLETED"!!0
1.3 P

C FOCAL=65 (V3D) 18 JUL '77

1.10 F X=1,10; "X="X,"i! 1,(X-6),1.3
1.20 T "THE FOR LOOP CAME HERE WHEN COMPLETED"!!0
1.30 P

Gn
X= 1.000
X= 2.000
X= 3.000
X= 4.000
X= 5.000
X= 6.000

The FOR loop came here when completed.

T 5

X0(6)= 6.000

When 'X' was equal to 6, control was transferred to line 1.3, without having seen a carriage return, the return command returned control immediately back to the 'FOR' loop, but execution was stopped in the 'FOR' loop, and control transferred to the next focal line (in this case 1.2).
FOCAL allows for certain special operations to be performed which transcend normal arithmetic operations. These operations are accomplished through the use of functions. A function in Focal always begins with the letter 'F' (hence variable names cannot begin with the letter). A function name, usually 3 characters, follows the 'F', a 'function' also has an 'argument' list which is enclosed in parentheses following the function name. Some functions require more arguments than others, depending upon the operation performed by the functions. If a function requires arguments, then they are placed inside the parentheses and are separated with commas. A function takes the arguments, performs a specific operation using them, and all focal functions return as their value a single number. A function may appear anywhere in an arithmetic expression that a number could appear. The arguments to a function may be any arithmetic expression, including other function values. Let's look at a few simple Focal functions first:

The 'fabs' function takes one numeric argument and the value it returns is the absolute value of the argument. Examples:

\[
\begin{align*}
\text{fabs}(3,1) & = 3.000 \\
2 + \text{fabs}(-3,1) & = 5.000 \\
\text{fabs}(8,1) & = 8.000 \\
\text{fabs}(2,5) * 2,1 & = 5.000
\end{align*}
\]
THE 'FINT' FUNCTION TAKES ONE NUMERIC ARGUMENT AND THE VALUE IT
RETURNS IS THE 'GREATEST INTEGER LESS THAN THE NUMBER'. THUS, THE
VALUE RETURNED WILL HAVE NO FRACTIONAL PART. THIS FUNCTION DOES NO
ROUNDING (SEE 'FINR' BELOW) ON THE ARGUMENT. EXAMPLES:

- FINT(3.75), 3.000
- FINT(3.99), 3.000
- FINT(-3.14), -4.000
- FINT(3/2), 2.000
- FABS(1+FINT(-3.14)), 3.000

THE 'FINR' FUNCTION TAKES ONE NUMERIC ARGUMENT, ROUNDS TO THE
NEAREST WHOLE NUMBER, AND THEN PERFORMS THE 'GREATEST INTEGER LESS THAN
THE NUMBER' OPERATION. EXAMPLES:

- FINR(3.75), 4.000
- FINR(3.99), 4.000
- FINR(-3.14), -4.000
- FINR(-3.14), -4.000
- FINR(-3.5), -4.000
- FINR(2.9), 3.000
- FINR(5/2), 3.000
IT IS SOMETIMES USEFUL (FOR GAMES, SIMULATION, ETC.) TO HAVE THE
CAPABILITY TO GENERATE PSEUDO-RANDOM NUMBERS. HOWEVER, WHEN DEBBUGING
THE PROCESS OF REMOVING MISTAKES) SUCH PROGRAMS, IT IS NICE TO GET THE
SAME SEQUENCE OF RANDOM NUMBERS EACH TIME WE RUN THE PROGRAM. WHEN
THE PROGRAM IS THEN WORKING, WE WOULD LIKE TO BE Able TO GET A DIFFERENT
SET OF RANDOM NUMBERS EACH TIME THE PROGRAM IS EXECUTED. THE 'FRAN'
FUNCTION TAKES A SINGLE NUMERIC ARGUMENT, AND ALWAYS RETURNS AS ITS VALUE
A PSEUDO-RANDOM FRACTION BETWEEN 0 AND 1. IF THE VALUE OF
THE ARGUMENT IS GREATER THAN ZERO, THEN THE RANDOM NUMBER GENERATOR
ROUTINE IS INITIALIZED TO GIVE A FIXED SEQUENCE OF RANDOM
NUMBERS, IF THE VALUE OF THE ARGUMENT IS LESS THAN ZERO, THEN THE
RANDOM NUMBER GENERATOR IS INITIALIZED TO GIVE A RANDOM
SEQUENCE (DIFFERENT EACH TIME) OF RANDOM NUMBERS. IF THE VALUE OF THE
ARGUMENT IS EQUAL TO ZERO, THEN THE NEXT RANDOM NUMBER FROM THE SEQUENCE
IS RETURNED. SOME EXAMPLES:

```
S FRAN(1)
```

THIS 'SET' COMMAND CALLS THE 'FRAN' FUNCTION WITH AN ARGUMENT
GREATER THAN ZERO, WHICH INITIALIZED THE RANDOM NUMBER GENERATOR TO A FIXED
(REPEATABLE) SEQUENCE. HOWEVER, THE VALUE RETURNED BY THE FUNCTION WAS
IGNORED, SINCE NO VARIABLE NAME SUBSTITUTION WAS INDICATED.
MORE EXAMPLES:

```
FOR I=1,101 I=FRAN(0),!
  0.496
  0.488
  0.497
  0.497
  0.249
  0.658
  0.284
  0.503
  0.695
  0.244
```

```
FOR I=1,101 I=FRAN(1),!
  0.533
  0.544
  0.361
  0.667
  0.743
  0.478
  0.239
  0.426
  0.414
  0.588
```
* FOR I=1,10: F R A N(I),! 
  0.329
  0.021
  0.347
  0.789
  0.630
  0.953
  0.141
  0.576
  0.642
  0.721

* G F R A N(1)

* F I=1,10: F R A N()+,!
  0.996
  2.488
  2.497
  2.647
  2.240
  2.658
  2.284
  0.533
  0.695
  3.244

* F I=1,10: F R A N()+,!
  2.533
  2.944
  2.561
  2.667
  2.743
  2.473
  2.239
  2.426
  2.414
  2.588

* F I=1,10: F R A N()+,!
  0.329
  2.021
  0.047
  2.789
  2.630
  0.953
  2.141
  0.576
  0.642
  0.721
NOTICE THAT THE SAME SEQUENCE OF RANDOM VALUES WAS OBTAINED AFTER AN "FRAN(1)" CALL WAS ISSUED.  MORE EXAMPLES:

\[ S \text{ FRAN}(1) \]
\[ S \text{ FRAN}(1) \]
\[ \text{IF } i=1,10 \text{ IT FRAN}(1),1 \]
0.257
0.534
0.495
0.331
0.423
0.195
0.904
0.058
1.000
0.111

A DIFFERENT SET OF RANDOM VALUES WAS OBTAINED AFTER CALLING "FRAN" WITH A NEGATIVE ARGUMENT.  RANDOM NUMBERS WITHIN ANY RANGE MAY BE OBTAINED BY MULTIPLYING AND/OR ADDING APPROPRIATE SCALING FACTORS TO THE FRACTION RETURNED BY "FRAN".  FOR EXAMPLE, IN ORDER TO OBTAIN RANDOM INTEGERS WITHIN THE RANGE 0-9:

\[ \text{IF } i=1,20 \text{ IT FINT(FRAN)*10),1} \]
7.000
3.000
2.000
9.000
6.000
7.000
7.000
0.000
9.000
7.000
0.000
0.000
1.000
8.000
4.000
3.000
8.000
4.000
5.000
4.000
SOMETIMES WE NEED TO BE ABLE TO OUTPUT ANY CHARACTER WE WANT
TO AN OUTPUT DEVICE, OR BE ABLE TO INPUT ANY CHARACTER WE WANT TO FROM
AN INPUT DEVICE. FOCAL HAS TWO SPECIAL FUNCTIONS FOR THIS PURPOSE. THERE
IS A THING CALLED ASCII (AMERICAN STANDARD CODE FOR INFORMATION
INTERCHANGE) WHICH ASSIGNS A NUMERIC VALUE TO EACH OF THE POSSIBLE
CHARACTERS (THERE ARE 128 OF THEM). FOR INSTANCE THE ASCII CODE FOR THE
CHARACTER 'A' IS 65, THE ASCII CODES CAN BE FOUND IN ANY
COMPUTER REFERENCE BOOK. THE 'FOUT' FUNCTION TAKES ONE NUMERIC ARGUMENT
IN THE RANGE 0-255, AND OUTPUTS THE ASCII CHARACTER WHICH HAS THAT NUMBER
AS ITS CODE NUMBER. THUS ANY CHARACTER CAN BE OUTPUT WITH AN 'FOUT' FUNCTION
IF THE PROGRAMMER SUPPLIES ITS ASCII CODE AS THE ARGUMENT. SOME EXAMPLES

S FOUT(65)

A

S FOUT(66) IT 1

B

F 1=256 FOUT(65+1)
ABCDEFHIJKLMNOPQRSTUVWXYZ

HAVE YOU EVER WONDERED HOW YOU WOULD TYPE OUT A "" (DOUBLE QUOTE)?
HERE ARE SOME THINGS THAT DON'T WORK:

""
""
""
""

HERE IS SOMETHING THAT DOES WORK (KNOWING THAT THE ASCII CODE NUMBER
FOR A DOUBLE QUOTE IS 34):

S FOUT(34)

F 1=1,11FOUT(34)

---------------
FOCAL ALLOWS US TO PRECEDE A SINGLE CHARACTER WITH A SINGLE QUOTE
MARK, AND THAT REPRESENTS A NUMBER WHOSE VALUE IS
THE ASCII CODE FOR THE SINGLE CHARACTER. THUS, WE DON'T REALLY HAVE TO
KNOW WHAT THE CODE FOR A CHARACTER IS:

```
T 'A',
65.000
T 'B',
66.000
T 'C',
34.000
```

F 1=0.25 IS FOUT('A+1')
ABCDEFGHIJKLMOQRSTUVWXYZ

THE 'FCHR' FUNCTION REQUIRES NO ARGUMENT, INPUTS ONE CHARACTER
FROM THE INPUT DEVICE (ANY CHARACTER), AND THE NUMERIC VALUE RETURNED
AS THE VALUE OF THE FUNCTION IS THE ASCII CODE NUMBER FOR THAT CHARACTER.
FOR EXAMPLE:

```
S = FCHR()
A
```

THE USER TYPED THE CHARACTER 'A' FROM THE KEYBOARD.

```
T S,
65.000
```

```
E A
F 1=1.1815 C(I)*FCHR()
W1 THERE!
```
THE USER TYPED THE CHARACTERS 'HI THERE!' FOLLOWED BY A CARRIAGE
RETURN (THE 10TH CHARACTER); THE ASCII CODES FOR THESE CHARACTERS
WERE STORED IN THE 'C' ARRAY, TO SEE THEM:

```
10: 0) = 48,000
20: 1) = 72,000
30: 2) = 73,000
40: 3) = 32,000
50: 4) = 84,000
60: 5) = 72,000
70: 6) = 69,000
80: 7) = 82,000
90: 8) = 69,000
A0: 9) = 33,000
B0: (10) = 13,000
```

TO WRITE THE CHARACTERS BACK OUT:
```
IF [1:10]IS FOUT(C(I))
HI THERE!
```

TO SEE ONLY THE FIRST 5 CHARACTERS:
```
IF [1:5]IS FOUT(C(I))
HI THE
```

FOCAL HAS MORE POWERFUL FACILITIES FOR THE MANIPULATION OF CHARACTERS
WHICH IS EXPLAINED IN DETAIL LATER. THE 'FOUT' AND 'FCWR' FUNCTIONS
ALLOW THE PROGRAMMER TO GET BY THOSE SEEMINGLY IMPOSSIBLE QUESTIONS
SUCH AS 'HOW CAN I INPUT/OUTPUT THIS STRANGE CHARACTER?'.
FOCAL has the ability to transfer information to, and obtain information from, various devices that the user may have attached to his computer system. The various hardware devices are assigned numbers by the FOCAL system. These are positive numbers in the range of 0-127.

The programmer may independently change which device FOCAL is inputting from or outputting to at any given instant, through the use of the 'FIOV' and 'FDOV' functions. The 'FIOV' function stores away the current device number of the current input device, then takes the single numeric argument as the device number of the device to make the currently active input device. The 'FDOV' function stores away the current device number of the current output device, then takes the single numeric argument as the device number of the device to make the currently active output device. Further input/output will take place using the new devices until either a different device is made current through a new call to 'FIOV'/'FDOV' or a 'RESTORE INPUT' (abbreviated 'IR') or 'RESTORE OUTPUT' (abbreviated 'OR') command is executed, which restores the input/output device back to what it was just prior to the last 'FIOV'/'FDOV'. In the following examples, assume that the current input device is device number 3, and the current output device is device number 5.

In order to write my FOCAL program to output device 8:

S FIOV(8) WR 0
C FOCAL-69 (V3D) 18-JUL-77

In order to output some numbers to output device 8:

F 1=1,1,1 S FIOV(8)IT 1,1,0
1.000 2.000 3.000 4.000 5.000 6.000 7.000 8.000 9.000 10.000
**F I=1,10;S F(0)IT I,1;R 0
1.200
2.200
3.200
4.200
5.000
6.000
7.000
8.000
9.000
10.000

*C TO INPUT 10 CHARACTERS FROM INPUT DEVICE 0:

*1,1 S FIOV(0)
*1,2 F I=1,10;S C(I)=CHR(I)
*1,3 R 1

C FOCAL=65 (V3D) 18-JUL-77

1.10 G FIOV(0)
1.20 F I=1,10;S C(I)=CHR(I)
1.30 R 1

**G
**ABDEFGHIJ

**T $**

I( 0) = 11,000
C( 1) = 69,000
C( 2) = 68,000
C( 3) = 67,000
C( 4) = 66,000
C( 5) = 69,000
C( 6) = 70,000
C( 7) = 71,000
C( 8) = 72,000
C( 9) = 73,000
C(10) = 74,000

**F I=1,10;S FOUT(C(I))
**ABDEFGHIJ

**ERASE ALL**
Sometimes it is necessary to 'initialize' a device before it can be used to transfer data. Some devices require it, others don't. For example, cassettes do require initialization to allocate buffer space for data storage, etc., but teletypes may not require any initialization in order to be used. (The teletype interface might, however), it is good focal programming practice to initialize all devices before data transfer takes place. This is accomplished using the 'FINI' and the 'FINO' functions. 'FINI' calls a device dependent routine within focal to initialize a given device for input. 'FINO' calls a device dependent routine within focal to initialize a given device for output. The device number of the device is the value of the single argument. Some devices must also be 'closed', or told that input/output is over, so that they can finish any incompleted transfers (buffered cassette i-o is an example). The device can be 'closed' by the use of the 'FCLI' function if the device was initialized for input, or the 'FCLD' function if the device was initialized for output. These functions accept a single numeric argument which is the device number of the device to 'close'. It is good focal programming practice to 'close' a device when all input/output to that device has been completed. Some examples:

```
 0                           0
 1.1 S FINO(1),FDDV(1),MIN FDL(1);I R 0
```

Focal=63 (VSD) 18-Jul-79

```
 1.10 S FINO(1),FDDV(1),MIN FDL(1);I R 0
```

The above sequence will initialize device number 1 for output. Set device number 1 as the current output device, write the entire focal program to the device, close the device, and restore the output device back to whatever it was before the 'FDDV'. If device 1 was a cassette, the entire focal program would have been stored on the cassette.

Erase all

...
THIS ALLOWS THE USER TO STORE FOCAL PROGRAMS AND DATA ONTO OTHER
DEVICES THAT MAY BE CONNECTED TO HIS COMPUTER. THERE IS ONE DEVICE
WHICH DOES HAVE SOME SPECIAL SIGNIFICANCE TO FOCAL: THAT IS THE
USER'S CONSOLE DEVICE (THE DEVICE THAT HIS CONSOLE KEYBOARD AND
OUTPUT DEVICE, TELETYPE, CRT, ETC. IS CONNECTED TO), ALL ERROR
MESSAGES ARE OUTPUT TO THE USER'S CONSOLE DEVICE. THE USER MAY
CHANGE HIS CONSOLE DEVICE TO BE ANOTHER DEVICE ON THE COMPUTER
SYSTEM WITH THE 'FCON' FUNCTION. THIS FUNCTION ACCEPTS A SINGLE
NUMERIC ARGUMENT AND THAT NUMBER (0-127) BECOMES THE NEW
CONSOLE DEVICE NUMBER. THAT DEVICE STAYS THE CONSOLE DEVICE UNTIL
CHANGED BY THE USER WITH ANOTHER 'FCON' CALL. A NEGATIVE ARGUMENT
TO 'FCON' DOES NOTHING, BUT RETURNS THE DEVICE NUMBER OF THE CURRENT
CONSOLE DEVICE. SOME EXAMPLES:

TO FIND OUT WHAT THE CURRENT CONSOLE DEVICE'S NUMBER IS:

T FCON(+1); 1 3.000

TO MOVE THE CONSOLE TO DEVICE 3

S FCON(3)

THE CURRENT CONSOLE DEVICE IS DEVICE 0, BECAUSE:

T FCON(+1); 1 0.000

TO GO BACK TO DEVICE 3 AS THE CONSOLE:

S FCON(3)
FOCAL ALLOWS THE PROGRAMMER TO MANIPULATE 'BYTE' OR 'CHARACTER' STRINGS, AND PROVIDES SEVERAL FUNCTIONS WHICH FACILITATE SUCH OPERATIONS. LET'S LOOK AT 'BYTE' STRINGS FIRST. UP TO THIS POINT, ONLY NUMERICAL INFORMATION HAS BEEN READILY MANIPULATED. EVEN THE 'FOUT' AND 'FONR' FUNCTIONS USED NUMBERS TO REPRESENT THE CHARACTERS. THE MAIN PROBLEMS ARE THAT THE 'FOUT' AND 'FONR' FUNCTIONS GIVE THE USER LITTLE FLEXIBILITY IN MANIPULATING SERIES OF CHARACTERS OR 'BYTE', AND TO STORE A CHARACTER AS A NUMBER IN A NUMERIC VARIABLE NAME TAKES ABOUT 7 TIMES THE AMOUNT OF COMPUTER MEMORY STORAGE THAN MORE OPTIMAL METHODS, THUS FOCAL ALLOWS THE USER TO DEFINE AND USE 'BYTE' OR 'STRING' VARIABLES, AS THEY ARE CALLED. A 'STRING' VARIABLE IS A SEQUENTIAL SERIES OF 'BYTE' STORED IN THE COMPUTER'S MEMORY. IN GENERAL, NUMBERS IN THE RANGE 0-255 MAY BE STORED IN EACH 'BYTE' POSITION. IF THAT NUMBER WERE TO BE AN ASCII CODE NUMBER, THEN A CHARACTER COULD ALSO BE STORED THERE. FOCAL DOES NOT CARE WHAT THE INFORMATION IS OR WHAT IT REPRESENTS. A 'BYTE' STRING IS JUST A SERIES OF NUMBERS IN THE RANGE 0-255. 'BYTE' STRINGS ARE GIVEN VARIABLE NAMES, JUST LIKE NUMERIC VARIABLE NAMES (A-0 FOR FIRST CHAR, BUT NOT F, 0-7 FOR SECOND CHAR, SEE EARLIER DISCUSSION OF VARIABLE NAMES), HOWEVER A 'B' IS ADDED TO THE NAME IN ORDER TO IDENTIFY THAT VARIABLE NAME AS REPRESENTING A STRING OF BYTES. THE SUBSCRIPT USED WITH THE BYTE VARIABLE DETERMINES WHICH BYTE IN THE STRING IS BEING REFERENCED (THE FIRST BYTE IS SUBSCRIPTED 0, THE SECOND 1, THE THIRD 2, AND SO ON). A BYTE STRING MAY HAVE UP TO 256 BYTES STORED IN IT. IF A BYTE STRING HAS NOT BEEN ASSIGNED A LENGTH, (SEE 'FILS' FUNCTION), THEN THE DEFAULT LENGTH OF 72 BYTES IS ASSIGNED TO THE FIRST TIME THAT IT IS REFERENCED. ALSO ALL BYTES IN A STRING ARE INITIALIZED TO THE ASCII CODE FOR A BLANK (32), WHEN FIRST INITIALIZED. HERE ARE SOME EXAMPLES WHICH ILLUSTRATE STRINGS IN THE MOST PRIMITIVE MANIPULATIONS. IF A STRING VARIABLE IS USED IN AN ARITHMETIC EXPRESSION, THEN THE VALUE OF THAT VARIABLE IS THE NUMERIC VALUE OF THE BYTE STORED THERE (0-255). IF A CHARACTER IS STORED THERE, THEN THE NUMBER WILL BE THE ASCII CODE NUMBER FOR THAT CHARACTER.
C SOME EXAMPLES:

S AS(0)=65

T S

AGS="A"

C REMEMBER THE ASCII CODE NUMBER 65 REPRESENTS THE CHARACTER 'A'.

S AS(1)=66

T S

AGS="AB"

S AS(2)="C"

T S

AGS="ABC"

T AS(1),!

66.000

T AS(2),!

67.000

S FOUT(AS(1))

S

F I=0.25 I S AS(1)="A+A+1"

T S

AGS="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

10(0)= 25.000
THE 'S' OPTION OF THE 'TYPE' COMMAND OUTPUTS THE BYTE STRING
ASSUMING THAT ASCII CHARACTER CODE NUMBERS ARE STORED IN EACH BYTE
POSITION. NOTE THAT 'AS' AND 'A$()' ARE THE SAME THING,
NAMELY THE FIRST BYTE IN THE STRING WHOSE NAME IS 'A' (OR 'A$'), SINCE
THEY ARE THE SAME. AGAIN REFER TO EARLIER DISCUSSION OF VARIABLE NAMES.

IF WE WANTED TO COPY THE CHARACTERS IN 'AS' INTO ANOTHER STRING,
SAVING 'AS', A CRUDE WAY TO DO THAT MIGHT BE:

F I=0,1:5 $$(())=A$$(())
T $  
A$$=ABCDEFGHIJKLMNOPQRSTUVWXYZ
I0; 0)=  12,000
B$$=ABCDEFGHIJKLMNOPQRSTUVWXYZ

WE COULD INPUT 10 CHARACTERS INTO 'AS' BEGINNING AT SUBSCRIPT 31

F I=3,1:5 A$$(())=FCHR()
HELLO, OVERI*
T $  
A$$=ABCDEFGHIJKLMNOPQRSTUVWXYZ
I0; 0)=  14,000
B$$=ABCDEFGHIJKLMNOPQRSTUVWXYZ
These are very crude manipulations of the characters. Let's now look at several focal functions which allow more convenient manipulation of character strings. The FISL function allows the length of a character string to be defined. The first time it is referenced, it allows the programmer to set aside in the computer's memory only the number of bytes he needs (programmer may need more, or less, than the default value of 72). The FISL accepts an arbitrary number of arguments in pairs. The first argument of the pair is a numeric value which is the number of bytes to set aside in memory for the specified string. The second argument of the pair is the name of the string variable. Any number of string variable's lengths may be initialized in one call to FISL. By just placing more argument pairs in the call, in order for the FISL to work, the call must be the first time that the specified string variable has been referenced in the focal program. Some examples:

ERASE ALL

S FISL(16, AS, 20, BS)

AS=""
BS=""

This call set aside 16 characters for 'AS' and 20 characters for 'BS'. All the bytes were set to contain the ASCII code for a space. The strings may be used now.

I=0, I<16 AS(I)='A'+1

T S

AS="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
BS="100"

100 = 16,000

So if the programmer only needs a string with 5 bytes in it, only 5 bytes need to be allocated. Once a string has been allocated, then it's length may not be changed until the variable list is erased.
IT IS USEFUL TO BE ABLE TO INPUT STRINGS FROM THE INPUT DEVICE.
WHATEVER KIND OF DEVICE IT MIGHT BE, THE 'FSTI' FUNCTION ALLOWS THE
INPUTTING OF CHARACTERS FROM THE INPUT DEVICE AND THEIR STORING INTO A
STRING VARIABLE. THE 'FSTI' FUNCTION HAS TWO MANDATORY ARGUMENTS, AND
AN OPTIONAL THIRD ARGUMENT. THE FIRST ARGUMENT IS THE MAXIMUM
NUMBER OF CHARACTERS TO INPUT. THE SECOND ARGUMENT IS THE
STRING NAME AND SUBSCRIPT POSITION TO START PLACING CHARACTERS
FROM THE INPUT DEVICE INTO THE STRING. THE THIRD ARGUMENT, IF SUPPLIED,
IS AN ASCII CODE NUMBER FOR A SINGLE CHARACTER. THIS CHARACTER
IS CALLED THE 'TERMINATION CHARACTER'. IF THE TERMINATION CHARACTER IS
READ FROM THE INPUT DEVICE, INPUT STOPS AND THE FUNCTION RETURNS. THE
TERMINATION CHARACTER IS NOT STORED INTO THE STRING, BUT THE VALUE
RETURNED BY THE 'FSTI' FUNCTION IS THE ACTUAL NUMBER OF CHARACTERS THAT
WERE TRANSFERRED FROM THE INPUT DEVICE AND STORED INTO THE STRING. IF
THE INPUT DEVICE IS THE CONSOLE DEVICE, THEN RUBOUT PROCESSING WILL TAKE
PLACE, ALLOWING THE 'RUBOUT OUT' OF TypING MISTAKES.

SOME EXAMPLES:

* 1,1 E
1,2 S FISL(2B,AS,2B,BS)1S E=FSTI(2B,AS,7X)

C FOCAL=AS (V30) 18-JUL-79
1.1 E
1.2@ S FISL(2B,AS,2B,AT)1S E=FSTI(2B,AS,7X)

*GD
HELLO*

*TS

ARS="HELLO"
SBS=""
20(0)= 5,000
THE TERMINATION CHARACTER IN THE 'FSTI' WAS AN 'X', WHICH MEANT THAT
CHARACTERS WOULD BE TRANSFERRED FROM THE INPUT DEVICE (KEYBOARD IN THIS
CASE) INTO 'AS' BEGINNING AT SUBSCRIPT 0, UNTIL EITHER 28
CHARACTERS HAVE BEEN TRANSFERRED, OR UNTIL THE TERMINATION
CHARACTER 'X' HAS BEEN READ FROM THE INPUT DEVICE. THE VALUE RETURNED
BY THE FUNCTION IS THE ACTUAL NUMBER OF CHARACTERS TRANSFERRED (IN THIS
CASE 5), THE TERMINATION CHARACTER 'X' IS NOT STORED IN THE STRING.
MORE EXAMPLES

T S
A0S="ABCD"IJKLMN0PQRST"
B0S=""
B20(0) = 20,000
T S
A0S="" ""
B0S=""
B20(0) = 0,000

IN THIS CASE NO CHARACTERS WERE TRANSFERRED, BECAUSE THE TERMINATION
CHARACTER WAS INPUT AS THE FIRST CHARACTER READ.

T S
A0S="ZZZ"
B0S=""
B20(0) = 3,000


1,20 S FISL(20,AS,20,BS); S Z=FSTI(20,AS(2),$'X')

C FOCAL=69 (V3D) 15=JUL=77
1,10 E
1,20 S FISL(20,AS,20,BS); S Z=FSTI(20,AS(2),$'X')

GO
BOH X

$T $5

A-offset= BCD
B-offset= ""
Z.offset= 3,000

*/

C THE CHARACTERS WERE STORED IN 'AS' BEGINNING AT SUBSCRIPT POSITION
C 2 (THE THIRD CHARACTER). RUBOUT PROCESSING IS ALLOWED ONLY IF THE
C CONSOLE IS THE INPUT DEVICE. EXAMPLE:

GO\$\\\\\"THIS IS A TEST\\\\"

$T $5

A-offset= THIS IS A TEST
B-offset= ""
Z.offset= 14,000
THE 'FSTO' FUNCTION ALLOWS STRINGS TO BE TRANSFERRED EFFICIENTLY TO THE OUTPUT DEVICE. THE ARGUMENTS TO THE 'FSTI' AND 'FSTO' ARE IDENTICAL, EXCEPT THAT IN 'FSTO' CHARACTERS ARE READ FROM THE STRING BEGINNING AT THE SPECIFIED SUBSCRIPT POSITION, TRANSFERRED TO THE OUTPUT DEVICE, UNTIL EITHER THE MAXIMUM NUMBER HAVE BEEN OUTPUT, OR UNTIL THE TERMINATION CHARACTER HAS BEEN READ FROM THE STRING. THE TERMINATION CHARACTER IS NOT SENT TO THE OUTPUT DEVICE. THE VALUE RETURNED BY 'FSTO' IS THE ACTUAL NUMBER OF CHARACTERS TRANSFERRED TO THE OUTPUT DEVICE.

NOTE: IF THE FIRST ARGUMENT TO AN 'FSTI' OR 'FSTO' IS NULL (A COMMA, BUT NOTHING BEFORE IT), THEN THE MAXIMUM NUMBER IS INFINITE. IN THIS CASE A TERMINATION CHARACTER IS ADVISABLE.

SOME EXAMPLES:

AB$=" THIS IS A TEST "
BB$=" "
200(0)=14.000
*S 2=FSTO(4,AB$(2))
THIS=

FOUR CHARACTERS WERE TRANSFERRED FROM 'AB' BEGINNING AT SUBSCRIPT 2, TO THE OUTPUT DEVICE.

*S 2=FSTO(14,AB$(2))
THIS IS A TEST
*S AS$(16)="",
*T $

AB$=" THIS IS A TEST, "
BB$=" "
200(0)=14.000
*S 2=FSTO(AB$,')"
THIS IS A TEST
*T $

AB$=" THIS IS A TEST, "
BB$=" "
200(0)=16.000
*C THERE WAS NO MAXIMUM, SO CHARACTERS WERE TRANSFERRED FROM 'AB' TO THE OUTPUT DEVICE UNTIL THE TERMINATION CHARACTER ',' WAS READ FROM THE STRING. THE TERMINATION CHARACTER WAS NOT OUTPUT, AND THE VALUE RETURNED BY THE FUNCTION (16) WAS THE NUMBER OF CHARACTERS ACTUALLY OUTPUT.
*C THE 'FSTI' AND 'FSTO' FUNCTIONS ARE VERY EFFICIENT TIME-WISE
*C AND SHOULD BE USED FOR INPUT AND OUTPUT OF CHARACTER STRINGS WHENEVER POSSIBLE
SOMETIMES IT IS USEFUL TO LOOK FOR A CERTAIN GROUP OF CHARACTERS
TO SEE IF THEY ARE CONTAINED WITHIN A STRING OF CHARACTERS. LET
US ASSUME WE HAVE THE FOLLOWING STRING VARIABLES:

S FSTI1,AS1,'-
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID,

S FSTI1,BS1,'-
MEN.

T S

NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID

THE CHARACTERS BETWEEN SUBSCRIPTS 0 AND 2 INCLUSIVE ('MEN') IS 'BS'.
CERTAINLY ARE A SERIES OF CHARACTERS. IF WE WANT TO HAVE FOCAL
SEARCH 'AS' (0-71), LOOKING FOR THE CHARACTERS 'MEN', WE WOULD USE THE
'FSLK' FUNCTION. EXAMPLE:

Z=FSLK(0$2(1),8572775874585748567)

T S

NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID

MEN

29,308
THE VALUE RETURNED BY THE FUNCTION IS THE SUBSCRIPT IN 'AS' WHERE THE
CHARACTERS 'MEN' WERE FOUND. THE 'FSLK' FUNCTION REQUIRES TWO PAIRS
OF ARGUMENTS, THE FIRST PAIR DEFINES THE BEGINNING AND ENDING POINT OF
SOME CHARACTER STRING WHICH IS A SUBSET OF A STRING (IN THIS CASE
'BS' FROM BYTE 0 THRU BYTE 2). THE SECOND PAIR
DEFINES ANOTHER BEGINNING AND ENDING POINT OF THE CHARACTER STRING TO
SEARCH (IN THIS CASE 'AS' FROM BYTE 2 THRU BYTE 71). THE SECOND STRING
IS SEARCHED, LOOKING FOR THE FIRST STRING TO BE FOUND SOMEWHERE WITHIN IT.
IF THE FIRST STRING IS FOUND, THEN THE VALUE RETURNED BY THE FUNCTION
IS THE SUBSCRIPT OF WHERE THE MATCH WAS ENCOUNTERED, IF IT WAS NOT
FOUND, THEN THE VALUE -1 IS RETURNED AS THE VALUE OF THE FUNCTION.
THUS ONE CAN INPUT A STRING FROM THE KEYBOARD, CHECK TO SEE IF IT IS
ONE OF A KNOWN SERIES OF WORDS, AND PROCEED ACCORDINGLY.
MORE EXAMPLES:

QS Z=FSLK(8S,8S(2),AS(28),AS(43))
TS

AQ2S="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
BQ2S="MEN
B0(3) = 29,000

QS Z=FSLK(8S,8S,AS,AS(71))
TS

AQ2S="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
BQ2S="MEN
B0(3) = 13,000

QS THE 'M' WAS LOCATED AT CHARACTER SUBSCRIPT 13 IN 'AS'.

QS Z=FSLK(8S,8S(2),AS(30),AS(43))
TS

AQ2S="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
BQ2S="MEN
B0(3) = -1,000

QS THE CHARACTER STRING 'MEN' WAS NOT LOCATED WITHIN 'AS' SUBSCRIPTS

QS 30-42,
WE WILL NOW LOOK AT A VERY POWERFUL FACILITY OF FOCAL, THE ABILITY
TO DO INPUT AND OUTPUT TO STRING VARIABLES AS IF THEY WERE HARDWARE.
INPUT AND OUTPUT DEVICES. WE MAY SET A STRING VARIABLE (BEGINNING AT
A CERTAIN SUBSCRIPT) TO BE OUR CURRENT INPUT DEVICE OR OUR CURRENT OUTPUT
DEVICE WITH THE 'FIDV' OR 'FODV' FUNCTIONS (DESCRIBED EARLIER). WHEN THIS
IS DONE, INFORMATION WILL BE READ OR WRITTEN TO THE STRING
VARIABLE WITH ANY FOCAL COMMAND THAT DOES INPUT OR OUTPUT. SOME EXAMPLES:

S FODV(45)IT "THIS IS SOME INFORMATION":IR 0
"TS
ADS="THIS IS SOME INFORMATION

S FODV(10)IT "MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW":IR 0
"TS
ADS="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW

S FODV(4512)IT "XXX":IR 0
"TS
ADS="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW

C WE CAN PICK UP WHERE WE LEFT OFF IN THE STRING BY CALLING THE
C 'FIDV' OR 'FODV' WITH A NEGATIVE ARGUMENT:
S FODV(-1)IT "YYY":IR 0
"TS
ADS="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW

S FODV(-1)IT "THIS IS NEAT!":IR 0
"TS
ADS="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW

S FODV(-1)IT "THIS IS NEAT!":IR 0
"TS
ADS="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW

* F0DV(B$)IT %1,2+21R 0
* T S
* A0S="MXXXYY THIS IS NEATIB, IT'S FLEECE WAS WHITE AS SNOW
* B0S="A"
* S F0DV(B$)IT 2+31R 0
* T S
* A0S="MXXXYY THIS IS NEATIB, IT'S FLEECE WAS WHITE AS SNOW
* B0S="A 8"

* C IT IS NOT NECESSARY IN FOCAL TO HAVE SPECIAL FUNCTIONS TO CONVERT
* C NUMBERS TO CHARACTERS AND CHARACTERS TO NUMBERS:
* C
* S DS(1)=',IS BS(3)='.
* T S
* A0S="MXXXYY THIS IS NEATIB, IT'S FLEECE WAS WHITE AS SNOW
* B0S="4,8;"
* C WATCH THIS!
* S FIDV(B$)ASK X,YIR I
* T S
* A0S="MXXXYY THIS IS NEATIB, IT'S FLEECE WAS WHITE AS SNOW
* B0S="4,8;" X0(2)=4 Y0(2)=8
*C CONSIDER THIS:

*Erase all

*S FMOV(A$) IT "SOME DATA IN A STRING" IR O
*S T S

A$ = "SOME DATA IN A STRING"

*C To copy 'AS' into 'BS'!

*S FMOV(A$), FSTI(72, BS) IR 1
*S T S

BS = "SOME DATA IN A STRING"

*C This sequence sets 'AS' as the current input device; and then the
*C FSTI inputs 72 characters from the current input device and stores them
*C into 'BS', beginning at subscript 0.

*C CONSIDER THIS:

*S FMOV(CS) IT "1*X", I IR O
*S T S

CS = "SOME DATA IN A STRING"

CS = "1*X"
**C** NOTICE THAT SOME CHARACTERS '1+X.1' HAVE BEEN PLACED IN 'CS' AND THAT
**C** A CARRIAGE RETURN WAS ALSO BEEN PLACED THERE (RIGHT AFTER THE '1')
**C** NOTICE THAT THE CHARACTERS IN 'CS' FORM A VALID FOCAL COMMAND SEQUENCE, THE
**C** SEQUENCE WOULD BE PERFECTLY VALID IF IT WERE TYPED IN BY THE
**C** USER OR STORED WITH A LINE NUMBER PRECEDING IT, WELL, YOU GUESSED IT, IT
**C** IS POSSIBLE TO STORE A VALID SEQUENCE OF FOCAL COMMANDS IN A STRING
**C** VARIABLE, TERMINATE IT WITH A CARRIAGE RETURN (IT MUST BE TERMINATED WITH
**C** THE CARRIAGE RETURN!!!), AND HAVE FOCAL PERFORM A 'DO' OF THE COMMANDS
**C** STORED IN THE STRING, WATCH!
**C**
**C** DO CS
  1
**C** T NS 031S X=x5
**C**
**C** DO CS
  6.000
**C** F X=x 91D CS
  1.000
  2.000
  3.000
  4.000
  5.000
  6.000
  7.000
  8.000
  9.000
  10.000
**C**
C THIS IS HEAVY STUFF. FOCAL (SINCE IT IS A PURE INTERPRETER) DOESN'T
C CARE WHERE IT GETS COMMANDS FROM. AS LONG AS THEY ARE A SERIES OF
C CHARACTERS, THUS FOCAL CAN READ COMMANDS FROM HARDWARE DEVICES, STRINGS,
C OR WHATEVER. FOCAL IS ALSO VERY COMPACT, FOCAL CAN BE TOLD A LOT IN
C A VERY LITTLE SPACE. I DIGRESS SLIGHTLY TO PRESENT A SHORT PROGRAM
C WHICH WILL INPUT A NUMBER FROM THE KEYBOARD AND OUTPUT IT'S BINARY
C REPRESENTATION. THIS PROGRAM USES A PROGRAMMING TECHNIQUE CALLED
C 'RECURSION'. (FOCAL IS FULLY RECURSIVE). PUTTING ON THE WIZARD HATT.
C
C A

*1.1 A !"NUMBER" NID 1,21G 1,1
*1.2 S D(I=1) = N-2*N#INT(N/2)+0 (-N) 1,21T D(I) S I=1
*111

#W C FOCAL=65 (V30) 18-Jul-77

1.10 A !"NUMBER" NID 1,21G 1,1
1.20 S D(I=1) = N-2*N#INT(N/2)+0 (-N) 1,21T D(I) S I=1

*60

NUMBER:5
101
NUMBER:4
101
NUMBER:11
101
NUMBER:111
110100
NUMBER:1024
1200000000
NUMBER:1023
111111111
NUMBER:1511
11111111
NUMBER:255
1111111
NUMBER:12
10
NUMBER:
7-19 = 1,18

A!"NUMBER" NID 1,21G 1,1

C

C IF YOU THINK THAT'S NEAT, REFER TO THE 'FSBR' USER DEFINED FUNCTION
C FACILITY, OR THE 'SOFTWARE PRIORITY INTERRUPT SYSTEM' ('FPIG') DESCRIBED
C LATER ON.
SOMETIMES IT IS USEFUL FOR THE PROGRAMMER TO DEFINE HIS OWN LINE OR GROUP AS A FUNCTION, THEN WHENEVER HE WANTS THAT PARTICULAR FUNCTION INVOKED, HE USES THE 'FSBR' FUNCTION TO Invoke THE LOCAL COMMAND LINE OR GROUP, ONE NUMERIC ARGUMENT CAN BE PASSED TO THE ROUTINE AND THE ROUTINE CAN RETURN A SINGLE NUMERIC VALUE FOR THE VALUE OF THE 'FSBR' FUNCTION, THE ARGUMENT IS PASSED IN A PARAMETER INDEPENDENT FASHION (HEAVY COMPUTER SCIENCE JARGON).

THERE ARE ACTUALLY TWO ARGUMENTS TO THE 'FSBR' FUNCTION: THE FIRST IS A LINE NUMBER OR GROUP NUMBER OF THE LINE OR GROUP TO 'DO' AS THE FUNCTION (YES, AN ARITHMETIC EXPRESSION CAN BE HERE). THE SECOND IS THE NUMERIC ARGUMENT TO BE PASSED TO THE FUNCTION.

THE PRECISE SEQUENCE IS AS FOLLOWS: THE CURRENT VALUE OF THE VARIABLE 'I' IS PUSHED ON THE STACK, THE VARIABLE 'I' IS SET EQUAL TO THE NUMERIC ARGUMENT PASSED TO THE FUNCTION (SECOND ARG OF 'FSBR'), A 'DO' IS PERFORMED OF THE SPECIFIED LINE OR GROUP (THE FIRST ARG OF 'FSBR'), WHEN THE 'DO' RETURNS, THE VALUE RETURNED BY THE FUNCTION IS THE CURRENT VALUE OF 'I', AND THE OLD VALUE OF 'I' IS RESTORED FROM THE STACK. VARIABLE NAMES CAN BEGIN WITH THE CHARACTER 'I', HENCE 'FSBR', 'I' MAY BE USED AS VALID VARIABLE NAMES IN LOCAL PROGRAMS. HOWEVER, BY CONVENTION, A LOCAL PROGRAM SHOULD ONLY USE 'I' VARIABLES IN A USER-DEFINED FUNCTION IN ORDER TO BE ABLE TO WRITE USER-DEFINED FUNCTIONS WHICH ARE INDEPENDENT OF CALLING ROUTINE. SOME EXAMPLES:

```
T 99.1  S = S/2

T I HAVE MADE A VERY SIMPLE FUNCTION WHICH WILL TAKE THE ARGUMENT AND DIVIDE IT BY TWO. I NOW CALL IT VIA 'FSBR':

T FSBR(99.1), 1

T 95.05

FOR I = 1 TO 10, FSBR(99.1), I
  1.000  0.900
  2.000  1.800
  3.000  3.600
  4.000  7.200
  5.000  10.800
  6.000  14.400
  7.000  21.000
  8.000  28.800
  9.000  36.000
 10.000 45.000
```

- PAGE 94 -

* C LET'S MAKE A FUNCTION WHICH CONVERTS DEGREES TO RADIANS:
  *
  * 99.1 S &=#3.1415926/180
  *
  C FOCAL=65 (V3D) 18-JUL-77
  99.10 S &=#3.1415926/180
  *
  *F X=3.45,36017 X,FSQR(99,X),1
  0.000  0.000
  45.000  0.785
  90.000  1.571
  135.000  2.356
  180.000  3.142
  225.000  3.927
  270.000  4.712
  315.000  5.498
  360.000  6.283
  *
  *F X=0.45,36017 X,X,5.05,FSQR(99,X),1
  0  0.00000
  45  0.78540
  90  1.57079
  135  2.35619
  180  3.14599
  225  3.92099
  270  4.71238
  315  5.49778
  360  6.28318
  *
  *7 5.93
  *
  *
* LET'S WRITE A USER DEFINED FUNCTION WHICH TAKES THE SQUARE
* ROOT OF THE ARGUMENT GIVEN IT. THIS FUNCTION USES THE COMMON
* 'NEWTON-RAPHSON' ITERATION.

* A

* S A, & A=2.43*0.00001
* I ((A+(A/2))/2)*2*ABS(A-A2*41/4))99.2

* W

C FOCAL=68 (V3D) 18-JUL-77
99.1 S A*&, A=2.43*0.00001
99.20 I ((A+(A/2))/2)*2*ABS(A-A2*41/4))99.2

* C  YES, THAT'S THE WHOLE THING! LET'S TRY IT,

* T %5.05
* T FSBR(99,49),! 7.00000
* T FSBR(99,2),! 1.41421

* FOR X=1,1,101 Y %5,X,%5.05,FSBR(99,X),!
  1  1.00000
  2  1.41421
  3  1.73205
  4  2.00000
  5  2.23607
  6  2.44949
  7  2.64575
  8  2.82843
  9  3.00000
 10  3.16228

* T %5.03

* C AND IF YOU DON'T LIKE MY SQUARE ROOT ROUTINE, JUST WRITE
* C YOUR OWN AND USE IT INSTEAD OF MINE!
THE 'FSBR' FUNCTION IS RECURSIVE (HEAVY COMPUTER SCIENCE JARGON).

AS ARE MOST FOCAL FUNCTIONS, THIS IMPLEMENTATION OF FOCAL DOES NOT
HAVE ANY INTRINSIC FUNCTIONS TO DO SUCH THINGS AS TRIGONOMETRIC
FUNCTIONS (SIN, COSINE, LOG, EXP, ARCTAN, ETC.); HOWEVER, THESE
FUNCTIONS CAN BE MADE AVAILABLE BY SIMPLY WRITING ROUTINES IN FOCAL
TO PERFORM THE NECESSARY CALCULATIONS, THEN CALL THEM FROM THE
APPLICATION PROGRAM WITH 'FSBR' CALLS, FOCAL ROUTINES TO DO ALL
THE COMMON TRIG FUNCTIONS (USING 'FSBR') ARE GIVEN IN AN APPENDIX.
ALSO A ROUTINE TO OUTPUT A NUMBER IN 'E' FORMAT (SCIENTIFIC NOTATION)
IS ALSO SHOWN THERE. IN MANY CASES, THESE ROUTINES ARE SMALLER IN
SIZE (BUT SLOWER) THAN THE EQUIVALENT ROUTINES WRITTEN IN ASSEMBLY
LANGUAGE. HERE IS AN EXAMPLE OF A RECURSIVE 'FSBR' FUNCTION TO CALCULATE
THE FACTORIAL OF A NUMBER. THE FACTORIAL OF N! IS DEFINED MATHEMATICALLY
TO BE N*(N-1)*(N-3)*...*3*2*1. I.E., THE PRODUCT OF ALL THE
INTEGERS UP THRU 'N'. THE EXAMPLE:

```
*ERASE ALL
99.1 I = (1+3)*99,21R
99.2 S = &F#FSBR(99, &-1)

C FOCAL=65 (V30) 10-JUL-77
```

```
99.10 I = (1+3)*99,21R
99.20 S = &F#FSBR(99, &-1)

*F X=7,-1,1.1T "THE FACTORIAL OF "X," IS "#FSBR(99, X),!" THE FACTORIAL OF 7.000 IS 5040.000
THE FACTORIAL OF 6.000 IS 720.000
THE FACTORIAL OF 5.000 IS 120.000
THE FACTORIAL OF 4.000 IS 24.000
THE FACTORIAL OF 3.000 IS 6.000
THE FACTORIAL OF 2.000 IS 2.000
THE FACTORIAL OF 1.000 IS 1.000
```
FOCAL has a powerful facility aimed at the experimenter and real-time user. A focal program can be interrupted by some external event (a door opening, a phone ringing, a burglar entering) and a 'DO' of a specified focal line or group performed, and control is automatically returned to the interrupted routine. Furthermore, the various interrupting devices can be assigned a priority, and the highest priority event will be the first serviced, the second highest priority will be the next serviced, and so on; Focal does not know (or care) what caused the event to happen, but deals with events as bits (binary digits) that are set in an 'event byte' stored in the computer's memory. The events correspond to bit positions in this byte from right to left (right is event 1, left is event 8), event 8 is the highest priority, and event 1 is the lowest. Whenever focal prompts with a '*' (has nothing to do), it disables the ability to be interrupted, thus, the focal programmer must enable focal to look at the event bits and interrupt the focal program when a given event (or group of events) happens. Someone must also write an assembly language routine to set the appropriate bits in the event byte when the particular event actually happens. The programmer uses the 'FPIC' function to manipulate the software priority interrupt system. The 'FPIC' function takes an arbitrary number of arguments, in pairs, and uses them in controlling the software interrupt system. The first argument of a pair is a software event number (1-8), the second argument is a focal line of group number (arithmetic expressions are allowed here) to 'DO' when the specified event bit gets set. The 'FPIC' call enables focal to check the event bit each time a new focal command is retrieved, and if the bit is set, a 'DO' of the specified line or group will be performed. When the 'DO' returns, focal will continue the interrupted statement. Focal also clears the event bit before it performs the 'DO'. If several events (bits) have been enabled with the 'FPIC' function, then focal will perform the 'DO' for the highest priority one first, the next highest priority, etc. The value returned by the 'FPIC' function is a number between 0 and 255 which will have bits set for each event that has been enabled. Thus, if events 2 and 3 have been enabled, a value of 5 will be returned. If the first argument of a pair is 8, then the second argument is a number 0-255 which is used to indicate which events are to be enabled. This allows the software interrupt system to be temporarily disabled (by 'FPIC(8,0)', and enabled again (in the example by 'FPIC(8,9)'). If the first argument is 8 and the second is negative, then the 'FPIC' function does nothing but return the enable byte (as it always does). Note that all active event channels should be enabled with an 'FPIC' call which specifies the line or group to 'DO' first, before they are disabled/enabled later, since it is necessary for focal to store the line number or group number in internal tables.
WELL, AFTER THAT LONG-WINDED EXPLANATION, LET'S LOOK AT AN EXAMPLE. LET US SAY THAT A SWITCH CONNECTED TO A DOOR WILL GENERATE AN INTERRUPT TO THE COMPUTER AND A ROUTINE WILL SET EVENT BIT 2. ALSO, LET US SAY THAT A THERMOCOUPLE CIRCUIT CONNECTED TO THE ROAST IN THE OVEN WILL GENERATE AN INTERRUPT TO THE COMPUTER AND A ROUTINE WILL SET EVENT BIT 2 WHEN THE TEMPERATURE IN THE ROAST REACHES A CERTAIN VALUE. ALSO, LET US SAY THAT A PUSHBUTTON WILL GENERATE AN INTERRUPT AND SET BIT 7 WHEN THE BUTTON IS PUSHED.

HERE IS AN EXAMPLE FOCAL PROGRAM WHICH WILL ENABLE FOCAL TO SENSE THESE CONDITIONS, INTERRUPT THE PROGRAM (WHICH IS AN INFINITE LOOP), AND INFORM THE USER THAT THE EVENTS HAVE HAPPENED.

% 3
% E A
% #1.1 S FPIC((1,91,2,92,7,97))
% #1.3 S X=X+1/G 1.3

% #91.1 T "SOMEONE IS AT THE DOOR!" 1D 99
% #92.1 T "THE ROAST HAS REACHED TEMPERATURE!" 1D 99
% #97.1 T "INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM." 1!1Q
% #99.1 T " X=X,X!1

FOCAL=65 (V3D) 18-JUL-77

1.18 E
1.28 S FPIC((1,91,2,92,7,97))
1.38 S X=X+1/G 1.3

91.18 T "SOMEONE IS AT THE DOOR!" 1D 99
92.18 T "THE ROAST HAS REACHED TEMPERATURE!" 1D 99
97.18 T "INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM." 1!1Q
99.18 T " X=X,X!1

#0
SOMEONE IS AT THE DOOR: X=X 2897.000
THE ROAST HAS REACHED TEMPERATURE: X=X 3864.000
INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM,
THE PROGRAM ENABLED FOCAL TO 'DO' GROUP 91 IF EVENT 1 WAS SET,
92 IF EVENT 2 WAS SET, AND 97 IF EVENT 7 WAS SET. THE PROGRAM
THEN ENTERED AN INFINITE LOOP INCREMENTING THE VARIABLE 'X1', WHEN
SOMEONE OPENED THE DOOR, GROUP 91 WAS PERFORMED; WHEN THE ROAST
REACHED TEMPERATURE, GROUP 92 WAS PERFORMED, WHEN THE USER PRESSED
THE PUSHBUTTON, GROUP 99 WAS PERFORMED, AND THE PROGRAM WAS STOPPED.
LET'S SEE WHAT HAPPENS WHEN THE DOOR IS OPENED AND THE ROAST WAS
REACHED TEMPERATURE AT THE SAME TIME.

THE ROAST HAS REACHED TEMPERATURE! X= 1027.000
SOMEONE IS AT THE DOOR! X= 1027.000
INTERRUPT ON LEVEL 7. I'M STOPPING THIS PROGRAM.

NOTE THAT GROUP 92 WAS PERFORMED FIRST, SINCE THE ROAST IS
ASSOCIATED WITH A HIGHER EVENT NUMBER. HOWEVER, AS SOON AS
GROUP 92 RETURNED, THE DOOR INTERRUPT (GROUP 91) WAS PERFORMED
IMMEDIATELY (AS EVIDENCED BY THE FACT THAT 'X1' DID NOT GET INCREMENTED).
WHEN THE PUSHBUTTON WAS PRESSED, GROUP 97 WAS PERFORMED AND THE
PROGRAM WAS STOPPED. THE POSSIBLE USES OF THIS FACILITY ARE ALMOST
UNLIMITED.
HE WILL NOW LOOK AT A FEW REMAINING MISCELLANEOUS FUNCTIONS.

THE 'FCH' FUNCTION ALLOWS THE USER TO ENABLE/DISABLE THE AUTOMATIC
ECHOING OF CHARACTERS READ FROM THE INPUT DEVICE. 'FCH(0)' ENABLES
THE ECHOING, AND 'FCH(1)' DISABLES THE ECHOING.

THERE IS A FOCAL FUNCTION WHICH IS SPECIFIC TO THE CONSOLE
DEVICE. THIS FUNCTION IS IN FOCAL PRIMARILY BY POPULAR DEMAND,
SINCE IT IS USEFUL FOR GAME, ETC. THE 'FOUR' FUNCTION ALLOWS THE
PROGRAMMER TO POSITION THE CURSOR ON CRT TYPE TERMINALS (IF HE HAS A
CRT TYPE TERMINAL) TO A GIVEN ROW AND COLUMN ON THE SCREEN, WITHOUT
DISTURBING OTHER INFORMATION ON THE SCREEN. THE 'FOUR' FUNCTION
TAKES TWO ARGUMENTS, THE FIRST IS THE ROW NUMBER (R-N) TO MOVE TO,
The SECOND IS THE COLUMN NUMBER (C-N) TO MOVE TO. THE USER
MUST PATCH HIS CRT SPECIFIC ROUTINE INTO FOCAL SEE THE APPENDIX TO
FIND OUT HOW TO DO THIS.

FOR THOSE HACKERS THAT INSTRUCT SUCH THINGS, THE 'FHEM' FUNCTION
ALLOWS THE PROGRAMMER TO READ AND/OR WRITE INFORMATION FROM/TO
STORAGE LOCATIONS IN HIS COMPUTER'S MEMORY. THE 'FHEM' FUNCTION
IS ALWAYS GIVEN TWO ARGUMENTS, THE FIRST IS THE PAGE NUMBER (0-255)
AND THE SECOND IS THE LOCATION (0-255) WITHIN THAT PAGE. THESE
TWO VALUES FORM THE ADDRESS IN THE MEMORY, THUS TO ACCESS
MEMORY ADDRESS 025 (HEXADECIMAL) THE ARGUMENTS WOULD BE
1 (FOR THE PAGE NUMBER) AND 32 (FOR THE LOCATION WITHIN THAT PAGE).
(HEX = 16 DECEMAI). IF ONLY 2 ARGUMENTS ARE SUPPLIED, THE
VALUE RETURNED BY THE 'FHEM' FUNCTION IS THE DATA VALUE STORED IN THAT
MEMORY ADDRESS (A NUMBER 0-255). IF ANOTHER ARGUMENT FLOWS, THEN
THE VALUE OF THAT ARGUMENT (0-255) IS DEPOSITED (WRITTEN INTO) THE
MEMORY ADDRESS, AND THE VALUE RETURNED BY THE 'FHEM' FUNCTION IS THE
DATA VALUE THAT WAS STORED.

THESE BEFORE THE NEW VALUE WAS DEPOSITED.
SEVERAL DEPOSITS MAY BE MADE BY ONE CALL TO 'FHEM' BY GIVING SEVERAL
TRIPLER (GROUPS OF THREE) IN ONE CALL, EACH ONE SPECIFIES A PAGE NUMBER,
LOCATION IN THE PAGE, AND DATA TO STORED THERE. THE VALUE RETURNED
WILL BE THE DATA BYTE THAT WAS IN THE LAST LOCATION BEFORE THE NEW
BYTE TO STORED THERE. THIS IS USEFUL FOR THOSE THINGS OF MEMORY
WHERE SEVERAL THINGS MUST BE MESSIEБE BEFORE CONTROL IS RETURNED TO
THE FOCAL PROGRAM. (YES, IT IS POSSIBLE TO MAKE CHANGES TO THE FOCAL
SYSTEM WITH A FOCAL PROGRAM, YOU SHOULD BE VERY CAREFUL, AND KNOW
WHAT YOU ARE DOING.)

T FHEM(1,32):1
92.000
T FHEM(1,32,16),:
92.000
T FHEM(1,32),1
16.000
HERE IS A COMPLETE LIST OF ERROR CODES AND THEIR Meanings

-37  BAD OR MISSING ARGUMENT IN A STRING FUNCTION
-36  STRING VARIABLE REQUIRED HERE
-35  STRING VARIABLE NOT ALLOWED HERE
-34  I/O ERROR ON OUTPUT DEVICE
-33  ARGUMENT MISSING IN FUNCTION
-32  CURRENTLY NOT USED
-31  "WRITE" OF NON-EXISTENT GROUP
-30  UNRECOGNIZABLE FUNCTION NAME
-29  PARENTHESES ERROR IN FUNCTION
-28  "MODIFY" OF NON-EXISTENT LINE
-27  "DO" OF NON-EXISTENT GROUP
-26  "DO" OF NON-EXISTENT LINE
-25  SYNTAX ERROR IN "IF" OR "ON" COMMAND
-24  "ERASE" OF NON-EXISTENT LINE
-23  I/O ERROR ON INPUT DEVICE
-22  "WRITE" OF NON-EXISTENT LINE
-21  "GOTO" NON-EXISTENT LINE
-20  BAD LINE NUMBER ON INPUT
-19  UNKNOWN INTERRUPT REQUEST
-18  UNRECOGNIZABLE TRAP CODE
-17  RESET BUTTON PRESSED
-16  DEVICE NUMBER OUT OF RANGE
-15  USELESS "FOR" LOOP
-14  BAD TERMINATOR IN "FOR"
-13  NO "=" IN "FOR"
-12  BAD VARIABLE NAME
-11  FUNCTION ILLEGAL HERE
-10  NOT USED AT THIS TIME
-9  NOT USED AT THIS TIME
-8  FLOATING POINT OVERFLOW
-7  OPERAND MISSING -- EVAL
-6  PARENTHESES MISMATCH -- EVAL
-5  OPERATOR MISSING -- EVAL
-4  ILLEGAL LINE NUMBER
-3  UNRECOGNIZABLE COMMAND
-2  ILLEGAL GROUP ZERO USAGE
-1  LINE TOO LONG
TRIG FUNCTIONS IMPLEMENTED VIA 'FSBR' FUNCTIONS AS FOCAL Routines.

C FOCAL=65 (V3D) 18-JUL-77

93.01 C COS693 C SIN93.3
93.10 S (S-2.0)193.215 &/20 93.15 &2*8+2-11R
93.20 S &1-4+2/2*4+4/2&6/7201R
93.30 S &1.07068-4ID 03

94.01 C ASIN94 C ACOS94.3
94.10 S (S-2.0)194.215 &/6/(FSBR(99,1+6)+FSBR(99,1-6))1D 94.15 &2*81R
94.20 S &a&+4+3/6*.075&4+7/22.41R
94.30 C 94.31 &1.437796-8IR

95.01 C 4TAN
95.10 S (S-2.0)195.215 &X/(1+FSBR(99,4+21))1D 95.15 &2*81R
95.20 &a&1-4+2/3*4+8+4+7/7

96.01 C 4TAN
96.10 S (S-2.0)196.215 &X/20 96.15 &2*8/(1-2+2+961)R
96.20 S &a&1+4+3/3&5/7.5+7/7

97.01 C LOG
97.10 S (S-2.0)197.215 &X/FSBR(99,1)1D 97.15 &2*81R
97.20 &a&1-1/1&15 &2*+(S&+3/3&5/5&7/7)

98.01 C EXP
98.10 S (S-2.0)198.215 &X/20 98.15 &2*81R
98.20 S &a&1+4+2/2&4/6+4/2&6+5/720-6+5/720

99.01 C SQUARE ROOT
99.10 S &a&1&15 &21S &*1200801
99.20 S &a&1&11 (ABS(2+4-1&3)&a&3)99.3& &2*(S&+2)/21G 99.2

C FSBR(99,ARG) OUTPUTS ARG IN 'E' FORMAT

100.01 S &a&0
100.11 S (S-2.0)190.9-00
100.12 T =-05 .&a
100.20 S (S-1&1)190.15 (-S&SS9999999)190.73G 90.9
100.30 S &a&1&11S &8/10G 90.2
100.73 S &a&1&11S &8/10G 90.2
100.92 T &1.05;&8*&1.51R
I will now give some useful information for those people who have an assembly language expertise and want to hack things into it. These tips are by no means exhaustive, but cover the more common things.

There is a location on page zero labeled 'DELSPL' which Focal looks at to determine how it should handle rubber processing on the console device. Store a zero there if you have a device which is not a CRT (such as a teletype, dcewriter, etc.). If you have a CRT which will backspace the cursor when sent a 'backspace' character (ASCII code 18 octal), then store any non-zero value in 'DELSPL' to enable fancy CRT mode rubouts, where Focal 'eats' the character off the screen by sending the console the sequence of characters, 'backspace', 'space', 'backspace'.

The event byte for the software priority interrupt system is the byte stored at the label 'EVMSK'. Any interrupt routine can set bits in this byte to correspond to a focal event. (The LSB is event 1, the MSB is event 8.)

The 16 dispatch vectors must be set to point to the address 'INTSRV' so that focal error messages (which use the 'BRK' instruction) can be fielded properly. One easy way to do this is just put the code to set them in the console device initialization routine, which Focal calls the very first thing when it starts (see code at the label 'FOCAL').

To add I-O devices to focal, write an assembly language driver routine which knows how to talk to the device. The routine must have entry points to initialize the device for input (if an input device) and initialize the device for output (if an output device). The routine must also have entry points to close the device. If the device is an input device, there must be an entry point where Focal can call the driver in order to input an ASCII character from the device. Focal will call the routine with a 'JMP' instruction, the routine will return (via 'RTS') with the data byte in the accumulator, the 'C' bit must be clear if no errors were encountered on input, if the 'I' bit is set upon return from the routine, focal assumes that an input error occurred, and issues an error message, (see code at 'READ' in FOCAL). If the device is an output device, then there must be an entry point which Focal will call with the data byte to be output in the accumulator register. The routine will be called with a 'JMP' instruction and will return (via 'RTS') with the 'C' bit clear if no errors were encountered, and the 'I' bit set if an error occurred. The addresses of these entry points are placed in the device dispatch tables (see label 'DISP')). The relative position in the table determines the device number of that device. If more than five devices are inserted, the values of 'IDEV' and 'ODEV' must be updated. The only place they are referenced is at 'CHKDEV' and 'ICHKDEV'.

APPENDIX C
THE INITIAL DEVICE NUMBER OF THE CONSOLE DEVICE IS STORED IN
THE LOCATION 'CONDEV'. STORE A DIFFERENT NUMBER THERE IF YOU WANT YOUR
CONSOLE DEVICE TO BE SOMETHING OTHER THAN DEVICE NUMBER 0. THIS ONLY
Matters when Focal first starts up, since you can change to another
Device with the 'FCON' function.

ADDITIONAL COMMANDS MAY BE ADDED TO FOCAL BY PLACING THE FIRST
CHARACTER OF THE COMMAND (MUST BE DIFFERENT FROM EXISTING COMMANDS) IN
'COMADY' TABLE (THERE IS SPACE FOR HACKERS), AND THE ADDRESS OF THE
ROUTINE TO PROCESS THE COMMAND IN THE 'COMADY' AND 'COMADDY' TABLES.
LOOK AT THE CODE AT 'PROC1' TO SEE HOW FOCAL Dispatches TO COMMANDS.

NEW ASSEMBLY LANGUAGE FUNCTIONS MAY BE ADDED TO FOCAL BY ENCODING
THE FUNCTION NAME (USUALLY 3 CHARACTERS) INTO ITS 'HASH' CODE (SEE CODE
AT 'EFUN' TO DETERMINE HOW HASH CODE IS GENERATED) AND STORING THE HASH
CODE IN THE 'FUNCAB' TABLE. THE ADDRESS OF THE ROUTINE TO HANDLE THE
FUNCTION IS INSERTED INTO THE 'FUNADY' AND 'FUNADDY' TABLES. LOOK AT THE
CODE AT 'FUNC1', AND ANY OF THE STANDARD FOCAL FUNCTIONS TO SEE HOW THE
ROUTINE IS CALLED. NOTE: THE FIRST ARGUMENT OF A FUNCTION IS ALREADY
EVALUATED FOR YOU, AND IT'S VALUE IS STORED IN THE FLOATING-POINT
ACCUMULATOR, LOCATED ON PAGE ZERO AS 'FAC1'. IF YOUR FUNCTION IS TO
RETURN A VALUE, IT SHOULD STORE A NORMALIZED FLOATING-POINT NUMBER IN
'FAC1' PRIOR TO RETURNING (BY JUMPING TO 'FPOPJ').

NEED CAREFULLY THE WARNING PRINTED ABOVE SUBROUTINE 'PUSHJ'.

THE RANDOM NUMBER SEED GETS INITIALIZED TO RANDOMNESS BY LOADING
THE BYTE FROM PAGE ZERO LOCATION 'HASH'. SOME ROUTINE (GENERALLY KEYBOARD
INPUT ROUTINE) ON YOUR SYSTEM NEEDS TO OCCASIONALLY STORE JUNK IN THAT
LOCATION (SEE 'FRAK'). USUALLY THE KEYBOARD INPUT ROUTINE INCREMENTS
LOCATION 'HASH' AS IT'S WAITING FOR THE USER TO STRIKE A KEY ON THE
KEYBOARD, THUS THE VALUE WILL BE ESSENTIALLY RANDOM.

SOME KEYBOARDS SEND DIFFERENT CODES FOR THE 'ESCAPE' AND 'ALT'
ALTERNATE KEYS. FOCAL NORMALLY LOOKS FOR OCTAL CODE 33 AS THE SEARCH
OPTION IN THE 'MODIFY' COMMAND. IF YOU HAVE A STRANGE KEYBOARD, YOU
CAN PATCH THE VALUE AT 'MNXTC' AT WHATEVER ASCII CODE YOU WANT
IT TO BE.

IF YOUR HAVE A LOCAL COPY DEVICE FOR A CONSOLE (SOMETIMES; INCORRECTLY,
CALLED HALF-DUPLEX), THE CORRECT WAY TO HANDLE THIS IS IN YOUR DEVICE
SERVICE ROUTINE, BUT A QUICK HACK IS TO 'NOP' THE 'JSR PRINTC' LOCATED
AT 'READCE' ONE INSTRUCTION.

IF YOU IMPLEMENT A CURSOR ADDRESSING ROUTINE FOR YOUR CRT
CONSOLE DEVICE, PLACE THE ADDRESS OF THAT ROUTINE AS THE ADDRESS OF THE
'JSR CONCNR' IN THE 'FCUR' FUNCTION (SEE LABEL 'FCUR').