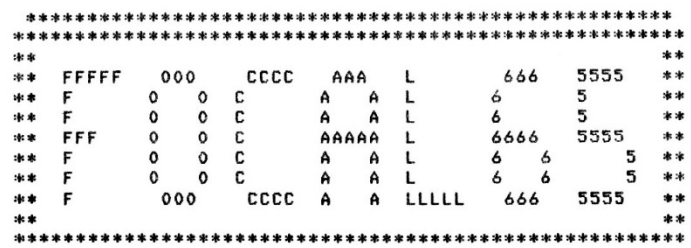


A USER'S GUIDE TO "FOCAL" FOR THE 6502

DISTRIBUTED BY: THE 6502 PROGRAM EXCHANGE
2920 WEST MOANA LANE
RENO, NEVADA 89509

***** FOCAL-65 PROGRAMMING LANGUAGE *****

***** FORMULATING ON-LINE CALCULATIONS IN ALGEBRAIC LANGUAGE *****

"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

T A B L E O F C O N T E N T S

C O M M A N D S A N D F U N C T I O N S

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
FISL	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

```
*
*C    FOCAL MUST BE GIVEN 'COMMANDS' IN ORDER TO ACTUALLY ACCOMPLISH
*C SOMETHING USEFUL TO THE USER.  THESE COMMANDS INSTRUCT FOCAL TO
*C PERFORM A SPECIFIC OPERATION OR SERIES OF OPERATIONS.  THE FOCAL
*C 'SYSTEM', WHICH RESIDES IN THE COMPUTER'S MEMORY, HAS BEEN
*C DESIGNED TO UNDERSTAND A SPECIFIC SET OF COMMANDS.  ANY COMMAND
*C THAT YOU GIVE TO FOCAL MUST BE ONE OF THESE SPECIFIC COMMANDS
*C THAT IT WAS DESIGNED TO RECOGNIZE.  IF YOU TRY TO GIVE
*C FOCAL OTHER COMMANDS, IT WILL NOT KNOW HOW TO INTERPRET THEM.
*C
*C   ONE OF THE MOST USEFUL COMMANDS IS THE 'TYPE' COMMAND.  THE
*C 'TYPE' COMMAND ALLOWS THE USER TO GIVE FOCAL AN ARITHMETIC
*C EXPRESSION, HAVE FOCAL EVALUATE IT, AND TYPE THE RESULTANT VALUE
*C ON THE USER'S OUTPUT DEVICE.  SOME SIMPLE EXAMPLES FOLLOW:
*
*TYPE 1+1
    2.000*
*
*C   LET US LOOK AT THE ABOVE 'TYPE' COMMAND.  THE USER ENTERS THE
*C COMMAND BY TYPING 'TYPE 1+1', AND THEN STRIKING THE 'CARRIAGE RETURN'
*C KEY ON HIS KEYBOARD.  THIS KEY IS SOMETIMES LABELED AS
*C 'RETURN'.  FOCAL DOES NOTHING WITH THE COMMAND UNTIL THE 'RETURN'
*C KEY IS STRUCK.  AT THAT POINT, FOCAL THEN TRIES TO INTERPRET THE
*C COMMAND AS ONE OF THOSE THAT IT HAS BEEN DESIGNED TO RECOGNIZE
*C (SUCH AS 'TYPE') AND THEN DOES THE APPROPRIATE THING THAT THE
*C COMMAND INDICATES TO DO.  IN THIS CASE, FOCAL WAS TOLD TO EVALUATE
*C THE ARITHMETIC EXPRESSION '1+1' AND TYPE THE RESULTANT VALUE TO
*C THE OUTPUT DEVICE.  IT DID THIS, SINCE THE VALUE '2,000' APPEARED
*C ON THE OUTPUT DEVICE.  AT THAT POINT, FOCAL HAD ACCOMPLISHED EVERYTHING
*C THAT THE COMMAND INDICATED, SO IT OUTPUTS THE '*' CHARACTER, WHICH
*C IS A PROMPT, TELLING THE USER THAT IT HAS NOTHING MORE TO DO.
*C FOCAL THEN WAITS FOR THE USER TO ENTER A NEW COMMAND.  A 'BLANK'
*C (SPACE BAR) MUST ALWAYS FOLLOW A FOCAL COMMAND, SINCE THIS IS
*C USED TO SEPARATE THE COMMAND NAME (SUCH AS 'TYPE') FROM THE REST
*C OF THE INFORMATION ON THE LINE (SUCH AS '1+1').  HOWEVER, THE COMMAND
*C NAME (SUCH AS 'TYPE') NEEDS ONLY TO BE EXPRESSED AS A SINGLE
*C CHARACTER (IN THIS CASE 'T') IN ORDER FOR FOCAL TO UNDERSTAND
*C WHICH COMMAND IT IS.  SOME MORE EXAMPLES FOLLOW!
*
*T 1+1
    2.000*
*T 2+3
    5.000*
*T 2+2+4
    8.000*T 1+1
    2.000*T 2+3
    5.000*T 2+2+4
    8.000*
*
```

```
*
*C   AS YOU CAN SEE, IF I JUST STRIKE THE 'RETURN' KEY, WITHOUT
*C TYPING A COMMAND OF SOME KIND, THEN FOCAL HAS NOTHING TO DO, AND
*C SIMPLY PROMPTS AGAIN WITH THE '*'.  SOME MORE EXAMPLES OF THE
*C 'TYPE' COMMAND:
*
*T 12.5+3.2
   15.700*
*
*T 10-7.5
    2.500*
*
*T 3*4
   12.000*
*
*T 3/4
    0.750*
*
*T 2*2*2
    8.000*
*
*T 2↑3
    8.000*
*
*C     AS YOU CAN SEE, SEVERAL DIFFERENT ARITHMETIC OPERATIONS CAN BE
*C PERFORMED BY FOCAL.  THESE ARE ADDITION, SUBTRACTION, DIVISION, MULTI-
*C PLICATION, AND EXPONENTIATION (RAISING TO A POWER).  THESE OPERATIONS
*C ARE INDICATED BY THE SYMBOLS + , - , / , * , ↑ , RESPECTIVELY.
*C WHEN THESE OPERATIONS ARE MIXED WITHIN A GIVEN EXPRESSION, THERE
*C IS A HIERARCHY RULE WHICH FOCAL USES TO DETERMINE THE ORDER IN WHICH
*C IT IS TO PERFORM THE OPERATIONS.  FOCAL WILL PERFORM ANY EXPONENTIATION
*C FIRST (↑), THEN ANY MULTIPLICATION (*), THEN ANY DIVISION (/), THEN
*C ANY SUBTRACTION (-), AND FINALLY ANY ADDITIONS (+).  SOME EXAMPLES
*C ILLUSTRATING THIS RULE FOLLOW:
*
*T 1+3*4
   13.000*
*
*T 2-3/4
    1.250*
*
*T 3-2
    1.000*
*
*T 2-3
   -1.000*
*
*T -2-3/4
   -2.750*
*
*T 3*5/2
    7.500*
*T 3*5/2↑2
    9.500*
*
*T 24/3*4
    2.000*
```

```
*
*C     THE USER MAY INDICATE THAT A CERTAIN GROUP OF OPERATIONS IS
*C TO BE PERFORMED FIRST.  HE DOES THIS BY ENCLOSING THAT GROUP OF
*C OPERATIONS WITHIN PARENTHESES.  THERE MAY BE MORE GROUPS ENCLOSED
*C WITH PARENTHESES WHICH ARE CONTAINED WITHIN A GROUP ALREADY ENCLOSED
*C WITHIN PARENTHESES.  IN THIS CASE, FOCAL WILL PERFORM THE
*C OPERATIONS WITHIN THE MOST DEEPLY NESTED (INNERMOST) PARENTHESES
*C FIRST, THEN THOSE IN THE NEXT OUTER, AND SO ON, UNTIL THOSE AT THE
*C OUTERMOST LEVEL ARE DONE LAST.  HERE ARE SOME EXAMPLES TO CLARIFY
*C THIS RULE!
*
*T 1+1
    2.000*
*
*T (1+1)
    2.000*
*
*T 24/3*4
    2.000*
*
*T (24/3)*4
   32.000*
*
*T 1+(2*(3*4))
   15.000*
*
*T 2↑3
    8.000*
*
*T 2↑(-3)
    0.125*
*
*T 1/8
    0.125*
*
*T 1+(2*(3*4)+3*(4/2))
   21.000*
*
*
*C     MESSAGES MAY ALSO BE OUTPUT BY USING THE 'TYPE' COMMAND.
*C BY SIMPLY ENCLOSING A SERIES OF CHARACTERS INSIDE OF DOUBLE
*C QUOTATION MARKS ("), THE 'TYPE' COMMAND WILL OUTPUT THE
*C SERIES OF CHARACTERS JUST AS THEY APPEAR INSIDE THE QUOTATION
*C MARKS.  SOME EXAMPLES!
*
*
*T "HI THERE"
HI THERE*
*
*T "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*
*
*T "ANY SERIES OF CHARACTERS"
ANY SERIES OF CHARACTERS*
*
```

```
*
*C      THE USER MAY INSTRUCT THE 'TYPE' COMMAND TO PERFORM SEVERAL
*C FUNCTIONS BY SEPARATING EACH FUNCTION FROM THE NEXT WITH
*C 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    3.500*
*
*C      AS YOU CAN SEE, THIS CAPABILITY ALLOWS OUTPUT FROM THE
*C COMPUTER TO BE MADE MORE LEGIBLE.  SOMETIMES IT IS
*C DESIRABLE TO HAVE CONTROL OVER THE LINE SPACING ON THE OUTPUT
*C DEVICE, IN ORDER TO MAKE THE OUTPUT APPEAR MORE LEGIBLE,  THERE
*C ARE SPECIAL FORMAT CONTROL CHARACTERS WHICH FOCAL RECOGNIZES WHEN
*C THEY APPEAR IN A 'TYPE' STATEMENT WHICH ALLOW THE USER TO DO THIS
*C KIND OF FORMATTING.  ONE SUCH CHARACTER IS THE EXCLAMATION MARK (!),
*C WHEN FOCAL ENCOUNTERS THIS CHARACTER IN A 'TYPE' STATEMENT, IT OUTPUTS
*C A CARRIAGE RETURN CHARACTER, FOLLOWED BY A LINE FEED CHARACTER, TO
*C THE OUTPUT DEVICE.  THIS CAUSES A RETURN TO THE BEGINNING OF THE
*C CURRENT LINE, AND AN ADVANCING TO THE NEXT LINE ON THE OUTPUT
*C DEVICE.  THE POUNDS CHARACTER (#), WHEN ENCOUNTERED IN A 'TYPE'
*C STATEMENT, CAUSES A CARRIAGE RETURN CHARACTER TO BE OUTPUT, BUT
*C THAT'S ALL.  THE EFFECT IS THAT THE CARRIAGE IS RETURNED
*C BUT THE LINE IS NOT ADVANCED.  HENCE, ANY FURTHER OUTPUT WOULD OCCUR
*C ON THE SAME LINE, POSSIBLY OVERPRINTING EXISTING OUTPUT.  SOME
*C EXAMPLES FOLLOW:
*
*T 1+1,!
    2.000
*T 1+1,!,2+3,!
    2.000
    8.000
*T "THE ANSWER IS ",3*4,!,"THE VALUE OF THE DISTANCE IS",4+3,!
THE ANSWER IS    12.000
THE VALUE OF THE DISTANCE IS   64.000
*
*T 1+1,!!,2+3,!!
    2.000


    8.000

*T "HI",!,"   THERE",!
HI
   THERE
*
*T "HI",#,"   THERE",!
HI THERE
*
*T !!,"      X         Y",!,2+3,4/5,!!


      X         Y
    5.000     0.800
```

```
*
*C      THIS CAPABILITY ALLOWS FOR MAKING VERY READABLE OUTPUT.
*C
*C
*C      SOMETIMES IT IS USEFUL TO HAVE FOCAL REMEMBER THE RESULT OF AN
*C ARITHMETIC CALCULATION.  THIS RESULT MAY THEN BE USED LATER INSTEAD
*C OF HAVING TO RE-DO THE CALCULATION OVER AGAIN.  THIS CAPABILITY IS
*C ACCOMPLISHED THROUGH THE USE OF 'VARIABLE NAMES'.  A NAME
*C MAY BE ATTACHED TO THE RESULT, OR PARTIAL RESULT, OF AN ARITHMETIC
*C CALCULATION.  IF THE NAME IS USED LATER IN SOME OTHER CALCULATION,
*C THEN THE VALUE ASSOCIATED WITH THAT NAME IS SUBSTITUTED IN PLACE OF
*C THE NAME.  IN FOCAL, A VARIABLE NAME MUST BEGIN WITH A LETTER OF
*C THE ALPHABET (A-Z), BUT MAY NOT BEGIN WITH THE LETTER 'F' (THIS LETTER
*C HAS A SPECIAL SIGNIFICANCE, DISCUSSED LATER).  A VARIABLE NAME MAY ALSO
*C HAVE AN OPTIONAL SECOND CHARACTER, WHICH MUST BE A DIGIT IN THE RANGE
*C OF 0-7.  WHEN THE OPTIONAL DIGIT IS OMITTED, THEN '0' IS
*C ASSUMED AS THE DIGIT.  THUS, 'A3', 'B7', 'C', 'Z2', AND 'R' ARE
*C VALID NAMES, WHILE SUCH NAMES AS, 'F2', 'Q8', AND 'AB' ARE NOT VALID
*C NAMES FOR VARIABLE QUANTITIES.  THE TERM 'VARIABLE' IS USED, BECAUSE,
*C AT SOME LATER TIME, THE SAME NAME MAY BE ASSOCIATED WITH A NEW QUANTITY,
*C THUS A NAME'S VALUE MAY 'VARY', FROM TIME TO TIME.  SOME EXAMPLES
*C WILL HELP TO CLARIFY THIS!
*
*
*T 1+1,!
   2.000
*T X=1+1,!
   2.000
*T X,!
   2.000
*T X+1,!
   3.000
*T X,!
   2.000
*T X=X+1,!
   3.000
*T X,!
   3.000
*T 2+3,!
   5.000
*T Y=2+3,!
   5.000
*T Y,!
   5.000
*T X,Y,!
   3.000    5.000
*T X+Y,!
   8.000
*T X=X+1,Y=Y+1,!
   4.000    6.000
*T X,Y,!
   4.000    6.000
*T 1+1/2,!
   1.500
```

```
*
*T X,Y,!
    4.000     6.000
*
*T A1=X+1,!
    5.000
*T A1,X,Y,!
    5.000     4.000     6.000
*
*T !!"A1=",A1," X=",X," Y=",Y,!!

A1=     5.000 X=     4.000 Y=     6.000

*
*T !!"A1=",A1,!,"X=",X,!,"Y=",Y,!!

A1=     5.000
X=      4.000
Y=      6.000

*
*C IN ORDER TO MAKE THE ABOVE MORE PRETTY!
*
*T !!"A1=",A1,!,"X =",X,!,"Y =",Y,!!

A1=     5.000
X =     4.000
Y =     6.000

*
```

```
*C      AS YOU CAN SEE, THE VALUES CURRENTLY ASSIGNED TO THE NAMES
*C 'A1','X', AND 'Y' WILL BE RETAINED BY FOCAL FOR USE IN LATER EXPRESSIONS.
*C ALSO NOTE THAT THE TWO NAMES 'A' AND 'A0' ARE ONE AND THE SAME.
*C SOME MORE EXAMPLES:
*
*
*T A=3↑2,!
  9.000
*T A,!
  9.000
*T A0,!
  9.000
*
*T A0=A0+1,!
 10.000
*T A0,A,!
 10.000   10.000
*
*T 2*(A↑1),!
 22.000
*
*T 2*A↑1,!
 21.000
*
*T 2*A-1,!
 19.000
*
*T A,!
 10.000
*
*T 2*(X=A↑1),!
 22.000
*T A,X,!
 10.000   11.000
*
*T 2*(A=A↑1),!
 22.000
*
*T A,!
 11.000
*
*T A1,X,Y,A,!
  5.000   11.000    6.000   11.000
*
*T A1=X=Y=A=1↑1,!
  2.000
*T A1,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 HE MAY USE THE FOCAL COMMAND 'SET'.  THIS
*C FOCAL COMMAND PERFORMS ANY VARIABLE NAME SUBSTITUTIONS, JUST LIKE THE
*C 'TYPE' COMMAND, BUT DOES NO 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=X+7,!
    8.000
*
*T X,Y,Z,!
    8.000    2.000    3.000
*
*S A1=X+1.5
*
*T A1,X,Y,Z,A,!
    9.500    8.000    2.000    3.000    2.000
*
*S 1+1
*
*T A1,X,Y,Z,A,!
    9.500    8.000    2.000    3.000    2.000
*
*
*C NOTE: SINCE NO SUBSTITUTION TOOK PLACE IN THE '$ 1+1' COMMAND ABOVE,
*C       THEN THE COMMAND SIMPLY EVALUATED THE EXPRESSION '1+1' AND DID
*C       NOTHING WITH THE RESULT!
```

```
*
*C A FEW MORE EXAMPLES:
*
*S A=X
*
*T A1,X,Y,Z,A,!
   9.500    8.000    2.000    3.000    8.000
*
*S A=Y=Z=A1
*
*T A1,X,Y,Z,A,!
   9.500    8.000    9.500    9.500    9.500
*
*
*C     VARIABLE NAMES (SUCH AS 'A1','X','Y','Z','A') 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.  IT
*C IS USEFUL, SOMETIMES, TO DEAL WITH SPECIFIC ITEMS OF A GIVEN NAME.
*C FOR INSTANCE, IF THE NAME 'C' REPRESENTED THE CHAIRS IN A ROOM, THEN
*C 'C(0)' ('C' SUBSCRIPT '0') MIGHT REPRESENT THE ZEROTH CHAIR IN THE ROOM.
*C (COMPUTER PROGRAMMERS SOMETIMES COUNT 0,1,2,... INSTEAD OF 1,2,3).
*C 'C(1)' 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),!
   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(0),!
   9.000    9.000
*
```

```
*
*C A USEFUL OPTION IN THE 'TYPE' COMMAND IS THE '$' OPTION.  WHEN A
*C '$' IS ENCOUNTERED IN A 'TYPE' COMMAND, FOCAL PRINTS ALL OF THE
*C VARIABLE NAMES, THEIR SUBSCRIPTS, AND THEIR CURRENT VALUE.
*C MORE EXAMPLES:
*
*T $

X@( 0)=     9.000
Y@( 0)=     9.500
A1( 0)=     9.500
A@( 0)=     9.500
Z@( 0)=     9.500
X@( 1)=     5.000
*
*T !"MY VARIABLES AND THEIR VALUES ARE!",!,$,!!

MY VARIABLES AND THEIR VALUES ARE:

X@( 0)=     9.000
Y@( 0)=     9.500
A1( 0)=     9.500
A@( 0)=     9.500
Z@( 0)=     9.500
X@( 1)=     5.000


*C NOW IT BECOMES EVIDENT THAT X AND X@ ARE THE SAME.  ACTUALLY,
*C X, X@, AND X@(@) ARE ALL ONE AND THE SAME VARIABLE NAME.
*C MORE EXAMPLES:
*
*S X(@)=10,X(1)=9,X(2)=8,X(3)=7
*
*T $

X@( 0)=    10.000
Y@( 0)=     9.500
A1( 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
*
*ERASE
*
*T S

*
*C NOTE THAT THERE ARE NOW NO DEFINED VARIABLE NAMES.
*
*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
*
*T 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 N=5
*
*T 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
N0( 0)=    5.000
*
```

```
*
*S N=N-3
*T $

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
N0( 0)=     2.000
*
*
*T X(N),!
     8.000
*
*T X(2),!
     8.000
*
*T X(N+1),!
     7.000
*
*T X(N)+1,!
     9.000
*
*T X(N-1),!
     9.000
*
*T $

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
N0( 0)=     2.000
```

```
*
*
*S X(9)=X(6)+1
*T 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)=    5.000
N0( 0)=    2.000
*
*S X(9)=N-1
*
*T 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
N0( 0)=    2.000
*
*C OK, WATCH THIS CAREFULLY!
*
*T X(N),!
    8.000
*
*T X(X(N)),
    2.000
*
*T X(8),!
    2.000
*
```

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

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
N0( 0)=    2.000
*
*S J(1000)=20
*T $

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
N0( 0)=    2.000
J0(1000)=   20.000
*
*S J(2000)=40
*T $

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
N0( 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 K(-10)=1,K(-9)=2,K(-1)=9,K(-4000)=2222,K=3
*T $

K0(-10)=    1.000
K0(-9)=     2.000
K0(-1)=     9.000
K0(-4000)=  2222.000
K0( 0)=     3.000
*
*S K1(-10)=1,K1(-9)=2,K1(-1)=9,K1(-4000)=2222,K1=3
*T $

K0(-10)=    1.000
K0(-9)=     2.000
K0(-1)=     9.000
K0(-4000)=  2222.000
K0( 0)=     3.000
K1(-10)=    1.000
K1(-9)=     2.000
K1(-1)=     9.000
K1(-4000)=  2222.000
K1( 0)=     3.000
```

```
*
*C THIS SHOWS THAT THE VARIABLE NAMES 'K' AND 'KØ' ARE THE SAME, BUT
*C ARE DIFFERENT FROM 'K1'.
*C MORE EXAMPLES:
*
*E
*T $

*S N=4
*T $

NØ( Ø)=     4.ØØØ
*
*S M=1Ø
*T $

NØ( Ø)=     4.ØØØ
MØ( Ø)=    1Ø.ØØØ
*
*T B(1),B(2),B(3),!
    Ø.ØØØ     Ø.ØØØ     Ø.ØØØ
*
*T $

NØ( Ø)=     4.ØØØ
MØ( Ø)=    1Ø.ØØØ
BØ( 1)=     Ø.ØØØ
BØ( 2)=     Ø.ØØØ
BØ( 3)=     Ø.ØØØ
*
*
*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.
*
*
```

```
*
*C     NOTICE THAT WHEN NUMBERS ARE OUTPUT BY FOCAL, THEY ARE OUTPUT
*C IN A CERTAIN FORMAT.  SOMETIMES THE USER WOULD LIKE TO CHANGE THE
*C FORMAT IN WHICH FOCAL OUTPUTS NUMBERS.  THIS IS ACCOMPLISHED USING
*C THE '%' (PERCENT) OPTION OF THE 'TYPE' COMMAND.  THIS OPTION ONLY
*C ESTABLISHES THE FORMAT THAT SUBSEQUENT NUMBERS WILL BE OUTPUT IN, BUT
*C DOES NO ACTUAL OUTPUT ITSELF.  ONCE THE OUTPUT FORMAT IS SET, ALL
*C SUBSEQUENT NUMBERS WILL BE OUTPUT IN THAT FORMAT UNTIL THE FORMAT
*C IS CHANGED WITH ANOTHER '%' OPTION TO THE 'TYPE' COMMAND.  THE '%'
*C OPTION MAY BE INTERSPERSED WITH OTHER OPTIONS OF THE 'TYPE' COMMAND
*C JUST LIKE '$','!', AND '#'.  WHEN THE '%' IS ENCOUNTERED IN A 'TYPE'
*C COMMAND, A NUMBER OF THE FORM 'BB.AA' IS ASSUMED TO FOLLOW IT.
*C THE 'BB' (00-99) INDICATES HOW MANY DIGITS ARE TO BE OUTPUT BEFORE
*C THE DECIMAL POINT (THE INTEGER PORTION OF THE NUMBER OUTPUT).  THE
*C 'AA' (00-99) INDICATES HOW MANY DIGITS ARE TO BE OUTPUT FOLLOWING
*C THE DECIMAL POINT (THE FRACTIONAL PORTION OF THE NUMBER OUTPUT).
*C IF 'AA' IS ZERO, THEN NO DECIMAL POINT IS ACTUALLY OUTPUT.
*C THE ACTUAL NUMBER OF CHARACTERS OUTPUT DOES NOT INCLUDE THE DECIMAL
*C POINT, NOR THE '-' SIGN PRECEDING THE NUMBER (IF ITS NEGATIVE).
*C NOTE THAT '%5.03' MEANS FIVE DIGITS BEFORE THE DECIMAL POINT, AND
*C THREE DIGITS AFTER THE DECIMAL POINT.  '%5.3' OR '%5.30' MEANS
*C FIVE DIGITS BEFORE AND THIRTY DIGITS AFTER THE DECIMAL POINT.  THE
*C NUMBERS USED IN ACTUAL CALCULATIONS ARE MAINTAINED TO ABOUT
*C 6 DIGITS OF SIGNIFICANCE.  THEY MAY LIE WITHIN THE RANGE OF
*C 10↑(-38) TO 10↑(+38).  SOME EXAMPLES SHOULD HELP CLARIFY THE
*C '%' OPTION TO THE 'TYPE' COMMAND.
*
*
*T 1+1,!
   2.000
*T 99.!
   99.000
*T %2.03,1+1,!
 2.000
*
*T 99.!
99.000
*
*T 96.!,97.!,98.!,99.!
96.000
97.000
98.000
99.000
*
*T 8.!,9.!,10.!,11.!
 8.000
 9.000
10.000
11.000
*
*T 98.!,99.!,100.!,101.!
98.000
99.000
100.000
101.000
*
```

```
*
*C      NOTICE THAT FOCAL WILL ALWAYS OUTPUT THE INTEGER PORTION OF THE
*C NUMBER, EVEN THOUGH IT MIGHT NOT BE ABLE TO FIT IN THE NUMBER OF
*C DIGITS THE USER HAS ASKED FOCAL TO PLACE BEFORE THE DECIMAL POINT.
*C THIS AT LEAST ALLOWS THE USER TO SEE THE NUMBER, EVEN THOUGH THE
*C DECIMAL POINTS WILL NO LONGER LINE UP IN COLUMN DATA.
*C MORE EXAMPLES:
*
*T 6+6,!
12.000
*
*T %5.03
*
*C NO OUTPUT WAS DONE IN THIS CASE, BUT A NEW OUTPUT FORMAT WAS
*C SPECIFIED.  MORE EXAMPLES!
*
*T 6+6,!
   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 AWAY INTERNALLY HAS NOT BEEN CHANGED.
*C THE ROUNDING WA  ONLY DONE FOR THE OUTPUT OPERATION.  MORE EXAMPLES!
*
*SET X=2.3456
*
*T X,!
    2.346
*
*T %5.04,X,!
    2.3456
*
*T %5.02,X,!
    2.35
*
*T %5.01,X,!
    2.3
*
*T %5.0,X,!
    2
*
*T %5.04,X,!
    2.3456
*
```

```
*
*S X=2.78
*
*T X,!
    2.7800
*
*T %5.02,X,!
    2.78
*
*T %5.01,X,!
    2.8
*
*T %5.00,X,!
    3
*
*S X=4
*
*T %5.03,X,!
    4.000
*
*T %5.3,X,!
    4.000000000000000000000000000000
*
*T %5.03,X,!
    4.000
*
*
*C    IT IS NOT NECESSARY FOR THE VALUE FOLLOWING THE '%' TO BE A CONSTANT
*C (SUCH AS '5.03'), BUT CAN BE ANY ARITHMETIC EXPRESSION.  FOCAL
*C WILL EVALUATE THE EXPRESSION, REDUCE IT TO A NUMBER OF THE FORM
*C 'BB.AA' AND USE THAT VALUE AS THE FORMAT SPECIFIER.  SOME EXAMPLES:
*
*
*S X=3.02
*
*T X,!
    3.020
*
*C CURRENT OUTPUT FORMAT IS STILL '5.03' FROM ABOVE.
*
*T 1+1,!
    2.000
*
*T %X,1+1,!
   3.00
*
*C THE OUTPUT FORMAT IS NOW '3.02', BECAUSE THAT IS THE VALUE OF 'X'.
*C MORE EXAMPLES:
*
*S X=X+1
*
*T %X,1+1,!
    2.00
*
```

```
*
*T $

X0( 0)=   4.02
*
*C THE OUTPUT FORMAT IS NOW '4.02', BECAUSE THAT IS THE VALUE OF 'X'.
*
*C WATCH THIS ONE CAREFULLY!
*
*T %X=X+.01,1+1,!
   2.000
*
*T $

X0( 0)=   4.030
*
*
*C    THE OUTPUT FORMAT WAS SET TO 'X' (AFTER 'X' WAS INCREMENTED BY .01).
*C SO THE OUTPUT FORMAT IS NOW '4.03', WHICH MEANS THAT FOUR DIGITS ARE OUTPUT
*C BEFORE THE DECIMAL POINT, AND THREE DIGITS ARE OUTPUT AFTER THE DECIMAL
*C POINT.  MORE EXAMPLES!
*
*
*T 1
   1.000*
*
*T 1,!,2,!,3,!
   1.000
   2.000
   3.000
*
*T -1,!,-2,!,-3,!
  -1.000
  -2.000
  -3.000
*
*T %5.03,1+1,!
    2.000
*
```

```
*
*C      LET'S NOW LOOK AT AN IMPORTANT FEATURE OF FOCAL,  THE 'TYPE'
*C AND 'SET' COMMANDS ALLOW THE USER TO DO SOME USEFUL THINGS,  THE
*C USER MAY NEED TO DO A SEQUENCE OF 'TYPE' AND 'SET' COMMANDS IN ORDER
*C TO ACCOMPLISH A CERTAIN TASK.  THE USER CAN PLACE MORE THAN ONE
*C COMMAND PER LINE BY SEPARATING EACH COMMAND WITH A ';' (SEMI-COLON).
*C SOME EXAMPLES:
*
*
*S X=5,Y=6,Z=7;T X,Y,Z,!
    5.000     6.000     7.000
*
*T $

X0( 0)=     5.000
Y0( 0)=     6.000
Z0( 0)=     7.000
*
*S Z=X+Y;T Z,!;S Z=Z+1;T X,Y,Z,!
   11.000
    5.000     6.000    12.000
*
```

```
*
*C      ONE OF THE MOST USEFUL FEATURES OF ANY COMPUTER LANGUAGE IS THE
*C ABILITY TO STORE A SERIES OF COMMANDS FOR LATER EXECUTION.  THE
*C COMPUTER IS VERY GOOD AT EXECUTING A SEQUENCE OF INSTRUCTIONS (COMMANDS)
*C OVER AND OVER AGAIN.  THE USER MAY STORE A LINE OF FOCAL COMMANDS
*C AWAY FOR LATER EXECUTION.  THIS IS DONE BY TYPING A 'LINE NUMBER'
*C BEFORE THE ACTUAL LINE OF COMMANDS.  WHEN THE CARRIAGE RETURN IS
*C STRUCK, THE LINE WILL BE STORED BY FOCAL, BUT THE COMMANDS ON THE
*C LINE WILL NOT BE EXECUTED (AS OPPOSED TO IMMEDIATE EXECUTION IF THE
*C 'LINE NUMBER' IS LEFT OFF.  'LINE NUMBERS' IN FOCAL ARE COMPOSED OF
*C TWO PARTS, AND HAVE THE FORM 'GG.SS', WHERE 'GG' (01-99) IS
*C THE 'GROUP' THAT THIS LINE BELONGS TO, AND 'SS' (01-99) IS THE
*C 'STEP' WITHIN THE 'GROUP'.  WE WILL SEE THE SIGNIFICANCE OF BEING
*C ABLE TO 'GROUP' LINES TOGETHER AND REFER TO THEM AS A UNIT.  THUS A
*C 'LINE NUMBER' OF '2.35' INDICATES THAT THIS LINE IS 'STEP' NUMBER 35
*C IN 'GROUP' 2.  NOTE THAT '2.2' FOR A LINE NUMBER MEANS THAT THIS LINE
*C IS STEP TWENTY IN GROUP TWO, AND NOT STEP TWO.  '2.02' WOULD BE
*C USED TO INDICATE THAT THE LINE WAS STEP TWO IN GROUP TWO.  SOME
*C 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",!
*
*
*C NOTICE THAT THE COMMANDS ON THESE LINES WERE NOT EXECUTED, BUT THE
*C LINES WERE STORED AWAY BY FOCAL, SO THAT WE MAY EXECUTE THEM ANY
*C TIME WE DESIRE.  WHENEVER A STORED LINE IS ENTERED (OR CHANGED), THE
*C USER'S VARIABLE NAMES AND THEIR VALUES ARE ERASED FROM THE COMPUTER'S
*C STORAGE.  THERE ARE SEVERAL FOCAL COMMANDS WHICH ARE USEFUL
*C TO THE USER BECAUSE THEY ALLOW MANIPULATION OF STORED LINES.  WE WILL
*C BE LOOKING AT THEM INDIVIDUALLY AS NEEDED.  THE FIRST OF THESE COMMANDS
*C IS THE 'WRITE' COMMAND ('W') WHICH ALLOWS THE USER TO WRITE OUT
*C 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 WAS 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"!!!S X=1;Y=2,Z=3

*W 2.1;W 1.2;W 1.1

 2.10 T "THAT'S ALL FOLKS",!

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

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

*
*T !,"HERE ARE MY STORED LINES",!!;W

HERE ARE MY STORED LINES

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

 1.10 T "HELLO, THERE"!!!S 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"!!!S 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",!

*
```

```
*
*C   THIS ALLOWS THE USER TO LIST ANY LINE, GROUP, OR THE ENTIRE
*C PROGRAM.  A 'PROGRAM' IS A SERIES OF STORED LINES WHICH PERFORM SOME
*C FUNCTION OR TASK FOR THE USER.  WELL, WE HAVE CAREFULLY TYPED IN THE
*C ABOVE STORED PROGRAM, BUT HOW DO WE INSTRUCT FOCAL TO ACTUALLY
*C PERFORM THE COMMANDS THAT HAVE BEEN STORED AWAY?  WE CAN INDICATE THAT
*C FOCAL TRANSFER CONTROL (BEGIN EXECUTING STATEMENTS) TO ANY SPECIFIC
*C LINE BY THE USE OF THE 'GOTO' (ABBRIEVIATED 'G' OR 'GO') COMMAND.
*C THE 'GOTO' COMMAND MUST BE FOLLOWED WITH THE LINE NUMBER OF
*C THE STORED LINE THAT WE WANT TO TRANSFER CONTROL TO.  IF A LINE NUMBER
*C IS OMITTED, THEN CONTROL IS TRANSFERRED TO THE LOWEST NUMBERED
*C LINE THAT HAS BEEN STORED AWAY.  SOME EXAMPLES:
*
*W

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

 1.10 T "HELLO, THERE"!!;S 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",!

*GOTO 1.1
HELLO, THERE

THE VALUES OF X, Y, Z, ARE      1.000     2.000     3.000
THAT'S ALL FOLKS
*
*
*C   NOTE THAT CONTROL PASSES TO THE NEXT LINE IN SEQUENCE (UNLESS FOCAL
*C HAS BEEN TOLD OTHERWISE) UNTIL ALL LINES HAVE BEEN EXECUTED.  FOCAL
*C THEN HAS NOTHING MORE TO DO; SO IT PROMPTS WITH A '*', AND AWAITS A
*C NEW COMMAND FROM THE USER.  MORE EXAMPLES!
*
*G 1.2
THE VALUES OF X, Y, Z, ARE      1.000     2.000     3.000
THAT'S ALL FOLKS
*
*G 2.1
THAT'S ALL FOLKS
*
*S X=11,Y=12,Z=13;G 1.2
THE VALUES OF X, Y, Z, ARE     11.000    12.000    13.000
THAT'S ALL FOLKS
*
*GO
HELLO, THERE

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

```
*
*T $

X0( 0)=     1.000
Y0( 0)=     2.000
Z0( 0)=     3.000
*
*1.15 G 2.1
*
*W

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

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

 2.10 T "THAT'S ALL FOLKS",!

*GO
HELLO, THERE

THAT'S ALL FOLKS
*
*
*C   NOTE THAT THE NORMAL SEQUENTIAL DIRECTION OF STAMEMENT EXECUTION
*C WAS ALTERED WHEN THE LINE '1.15' WAS INSERTED.  WE WERE INTRODUCED TO
*C THE 'ERASE' COMMAND EARLIER.  HOWEVER, THERE ARE OTHER USES FOR THE
*C 'ERASE' COMMAND.  IF THE ERASE COMMAND IS FOLLOWED BY A SPECIFIC
*C LINE NUMBER (SUCH AS '1.2'), IT ERASES JUST THAT LINE FROM STORAGE.
*C IF IT IS FOLLOWED BY A GROUP NUMBER ONLY (SUCH AS '1'), THEN IT ERASES
*C ALL LINES THAT BELONG TO THAT GROUP.  IF IT IS FOLLOWED BY THE
*C WORD 'ALL', THEN IT ERASES ALL OF THE STORED LINES.  AS WE
*C HAVE SEEN BEFORE, IF THE 'ERASE' COMMAND IS FOLLOWED BY NO LINE OR
*C GROUP NUMBER, THEN IT ONLY ERASES THE USERS VARIABLE NAMES AND THEIR STORED
*C VALUES.  ALL FORMS OF THE 'ERASE' COMMAND REMOVE ANY VARIABLE NAMES
*C AND THEIR ASSIGNED VALUES.  SOME EXAMPLES:
*
*E 1.15
*
*W

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

 1.10 T "HELLO, THERE"!!!S 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",!

*
*2.2 T "THIS IS THE LIVING END!",!
*
```

```
*
*W

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

 1.10 T "HELLO, THERE"!!!S 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 FOLK$",!
 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"!!!S X=1;Y=2,Z=3
 1.20 T "THE VALUES OF X, Y, Z, ARE ",X,Y,Z,!

*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
*
```

```
*C   SINCE THERE WERE NO STORED LINES, THERE WAS NOTHING FOR FOCAL
*C TO PERFORM.  AT THIS POINT, IT WOULD BE APPROPRIATE TO DISCUSS THE
*C QUESTION "WHAT IF I MAKE A MISTAKE?".  FOCAL PROVIDES SEVERAL
*C MECHANISMS WHICH ASSIST THE USER IN CORRECTING MISTAKES.  SOME WILL
*C BE DISCUSSED HERE, OTHERS LATER.  "WHAT IF I MAKE A MISTAKE WHILE
*C TYPING COMMANDS TO FOCAL FROM THE KEYBOARD?".  SINGLE CHARACTERS
*C CAN BE 'RUBBED OUT' BY STRIKING THE 'RUBOUT' KEY (SOMETIMES LABELED
*C 'DELETE') ON THE KEYBOARD,  ONE CHARACTER TO THE LEFT WILL BE
*C 'FORGOTTEN' BY FOCAL FOR EACH TIME THE 'RUBOUT' KEY IS STRUCK.
*C SOME EXAMPLES:
*
*
*T "THIS IS A TXOR\\\EST",!!
THIS IS A TEST

*
*
*C   NOTICE THAT THE '\' CHARACTER IS ECHOED EVERY TIME THE 'RUBOUT' KEY
*C IS STRUCK (A FANCY RUBOUT MODE FOR CRTS IS AVAILABLE, WHERE THE
*C CHARACTER IS 'EATEN' OFF THE SCREEN).  THE ENTIRE LINE TO THE
*C LEFT CAN BE 'FORGOTTEN' BY STRIKING THE 'BACKARROW' KEY ON THE KEYBOARD.
*C AN EXAMPLE:
*
*
*T "XYZTY←T "THIS IS A TEST",!!
THIS IS A TEST

*
*C   WHEN THE 'BACKARROW' KEY WAS STRUCK, FOCAL FORGOT EVERYTHING TO THE
*C LEFT OF THE 'BACKARROW'.  THE USER MAY JUST CONTINUE TYPING THE NEW
*C CHARACTERS ON THE SAME LINE.  MORE EXAMPLES:
*
*
*T "THIS ARE\\\\ IS A TXC\\EST",!!
THIS IS A TEST

*S X=1,Y=2,Z=3,←S WW←S X=7;Y=8;Z=10!T X,Y,Z,!!
   7.000    8.000   10.000

*
*
*C   HERE SEVERAL 'RUBOUTS' AND 'BACKARROWS' WERE USED ON THE SAME
*C LINE.  (YOU WOULD NEVER MAKE THAT MANY MISTAKES, OF COURSE).
*
```

```
*
*C      "WHAT IF A MISTAKE IS FOUND INSIDE OF A LINE ALREADY STORED AWAY?",
*C  ONE APPROACH WOULD BE TO SIMPLY TYPE THE WHOLE LINE IN OVER AGAIN (FOCAL
*C  WILL REPLACE THE ONE STORED WITH THE NEW LINE).  BUT THIS PROCESS CAN
*C  BE VERY TEDIOUS IF ONLY ONE CHARACTER IS TO BE CHANGED IN A LONG LINE.
*C
*C
*C      FOCAL PROVIDES A FACILITY WITH THE 'MODIFY' COMMAND TO ALLOW THE
*C  USER TO MAKE CHANGES IN A STORED LINE WITHOUT HAVING TO RE-TYPE THE
*C  ENTIRE LINE.  IF THE USER TYPES THE 'MODIFY' COMMAND ('M'), FOLLOWED
*C  BY A SPECIFIC LINE NUMBER, THAT LINE IS OPENED FOR MODIFICATION.  FOCAL
*C  TYPES OUT THE LINE NUMBER (INFORMING THE USER THAT IT LOCATED THE LINE),
*C  THEN WAITS FOR USER INPUT.  WHENEVER THE MODIFY COMMAND IS WAITING FOR
*C  USER INPUT, THE USER HAS SEVERAL MODIFICATION OPTIONS.  THEY ARE:
*C
*C
*C      1.   SIMPLY TYPE IN ANY TEXT THAT THE USER WANTS INSERTED AT THAT POINT.
*C
*C      2.   DELETE ANY TEXT TO THE LEFT BY THE USE OF THE 'RUBOUT' AND/OR
*C           'BACKARROW' KEYS.
*C
*C      3.   SEARCH FOR (POSITION AFTER) A CHARACTER FURTHER TO THE RIGHT OF THE
*C           USER'S CURRENT POSITION ON THE LINE.  THIS IS DONE BY STRIKING
*C           THE 'ALTMODE' KEY (SOMETIMES LABELED 'ESCAPE'), FOLLOWED BY
*C           STRIKING THE CHARACTER THAT IS TO BE SEARCHED FOR.  IF THE CHARACTER
*C           IS LOCATED, THE LINE IS TYPED OUT UP TO THAT POINT AND FOCAL
*C           WAITS FOR FURTHER USER INPUT.  IF THE CHARACTER IS NOT LOCATED,
*C           THE ENTIRE REST OF THE LINE IS TYPED, AND MODIFICATION ENDS.
*C
*C      4.   THE USER MAY TRUNCATE A LINE (REMOVE ALL INFORMATION TO THE RIGHT)
*C           BY TYPING THE CARRIAGE RETURN KEY.  AT THIS POINT MODIFICATION ENDS.
*C
*C      5.   THE USER MAY END MODIFICATION BY STRIKING THE 'LINE FEED' KEY.
*C           THIS CAUSES THE REMAINDER OF THE LINE TO BE TYPED, AND THEN
*C           MODIFICATION ENDS.  ALL DEFINED VARIABLE NAMES AND THEIR VALUES
*C           ARE ALSO ERASED.
*C
```

```
*
*C   IF THE USER WISHES TO MODIFY THE LINE AGAIN, HE SIMPLY TYPES A NEW
*C 'MODIFY' COMMAND.  THIS PROCESS IS A LITTLE HARD TO DEMONSTRATE ON THE
*C PRINTED PAGE, BUT HERE ARE SOME EXAMPLES:
*C
*C
*W

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

*1.1 T "THIS IS MY PRGRAM",!
*W

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

 1.10 T "THIS IS MY PRGRAM",!

*GO
THIS IS MY PRGRAM
*
*M 1.1

 1.10 T "THIS IS MY PROGRAM",!
*
*GO
THIS IS MY PROGRAM
*W

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

 1.10 T "THIS IS MY PROGRAM",!

*
*
*C    WHAT THE USER DID, IN THE ABOVE 'MODIFY' COMMAND, WAS TO TYPE AN
*C 'ALTMODE', THEN THE CHARACTER 'R',  FOCAL THEN TYPED OUT THE LINE UP
*C TO THE NEXT 'R' ENCOUNTERED (THE 'R' IN 'PRGRAM'),  THE USER THEN
*C STRUCK THE 'O' KEY, WHICH JUST INSERTED AN 'O' AT THAT POINT, THEN
*C STRUCK THE 'LINE FEED' KEY IN ORDER TO RETAIN THE REMAINDER OF THE LINE
*C AS IS,  THAT REQUIRED 4 KEYSTROKES AS OPPOSED TO 26 REQUIRED TO RE-TYPE
*C THE ENTIRE LINE.
*
*
```

```
*
*C      SOMETIMES IT IS USEFUL FOR THE USER TO SUPPLY DATA VALUES FOR
*C SOME OF THE VARIABLES IN THE PROGRAM, RUN THE PROGRAM, THEN
*C SUPPLY DIFFERENT VALUES, RUN THE PROGRAM AGAIN, ETC,  THE USER CAN INPUT
*C NUMERICAL INFORMATION FROM THE INPUT DEVICE AND HAVE FOCAL STORE THAT
*C INFORMATION IN A VARIABLE NAME, JUST AS IF HE HAD USED THE 'SET' COMMAND
*C TO DO IT.  THE FOCAL COMMAND WHICH DOES THIS IS THE 'ASK' (OR 'A') COMMAND.
*C FOR EACH VARIABLE NAME APPEARING IN THE 'ASK' COMMAND, FOCAL WAITS FOR
*C AN ARITHMETIC EXPRESSION TO BE INPUT, REDUCES IT TO A SINGLE NUMERIC
*C VALUE, AND ASSIGNS THAT VALUE AS THE VALUE OF THE VARIABLE NAME.
*C EXAMPLE:
*
*E ALL
*W

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

*T $

*ASK X,Y,Z
12.34
27.312
5
*
*T X,Y,Z,!!
   12.340    27.312     5.000

*T $

X0( 0)=    12.340
Y0( 0)=    27.312
Z0( 0)=     5.000
*
*
*A X,Y,Z
1,2,3
*
*T $

X0( 0)=     1.000
Y0( 0)=     2.000
Z0( 0)=     3.000
*
*T X,Y,Z,!!
    1.000     2.000     3.000

*
```

```
*
*C   CERTAIN NON-NUMERIC CHARACTERS CAN SEPARATE THE EXPRESSIONS ON TYPE-IN
*C (SUCH AS COMMA, OR CARRIAGE RETURN).  ANY ARITHMETIC EXPRESSION CAN
*C BE INPUT, AS THESE EXAMPLES SHOW:
*
*T $

X0( 0)=     1.000
Y0( 0)=     2.000
Z0( 0)=     3.000
*
*A B,C
X+Y+Z
Z↑2+1
*
*T $

X0( 0)=     1.000
Y0( 0)=     2.000
Z0( 0)=     3.000
B0( 0)=     6.000
C0( 0)=    10.000
*
*ASK X.Y,Z,B.C
X+1
Y+1
Z+1,B+1,C+1
*
*T $

X0( 0)=     2.000
Y0( 0)=     3.000
Z0( 0)=     4.000
B0( 0)=     7.000
C0( 0)=    11.000
*
```

```
*
*C    THE ABOVE SEQUENCE INCREMENTED EACH OF THE VARIABLES BY ONE.  THE
*C 'ASK' COMMAND ALSO RECOGNIZES THE '$', '"','%','!', AND '#' OPTIONS JUST
*C THE SAME AS THE 'TYPE' COMMAND.  THIS ALLOWS THE PROGRAMMER TO ADD
*C THINGS WHICH MAKE THE USE OF THE PROGRAM MORE COHERENT.
*C SOME EXAMPLES:
*
*E ALL
*W

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

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

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

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

*GO
NOW PLEASE ENTER A NUMBER: 4

THE VALUE OF THE NUMBER SQUARED IS   16.000

*G
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

*M 1.2

 1.20 T !,"THE VALUE OF THE NUMBER SQUARED IS=",X*2,!!
*W

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

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

*GO
NOW PLEASE ENTER A NUMBER: 4

THE VALUE OF THE NUMBER SQUARED IS=   16.000

*M 1.1

 1.10 A !,"NOW PLEASE ENTER A NUMBER! ",X,!
*
```

```
*
*W

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

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

*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=   10.240

*T S

X0( 0)=    3.200
*
```

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

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

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

*W 1.2

 1.20 T !,"THE VALUE OF THE NUMBER SQUARED IS=",X↑2,!!

*W 1.3

?-22

*
*C (ERROR CODE 22 IS 'WRITE OF NON-EXISTENT LINE')
*
*E 2
*
*C (NO COMPLAINT HERE, THERE WAS NOTHING TO ERASE)
*
*999.1 T "HI",!

?-4

999.1 T "HI",!


*
*C (ERROR CODE 4 IS 'ILLEGAL LINE NUMBER').  GROUP NUMBERS CAN ONLY
*C BE WITHIN THE RANGE 01-99.
*
*
*HELLO THERE

?-3

HELLO THERE


*
*C (ERROR CODE 3 IS 'UNRECOGNIZABLE COMMAND')
*
```

```
*
*C      WE HAVE SEEN HOW TO TRANSFER CONTROL IN THE FOCAL PROGRAM TC 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 (ABBREVIATED 'I') 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 (V3D) 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 !,"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

*
```

```
*
*C   CONTROL IS TRANSFERED TO LINE '1.3' IF THE VALUE OF 'X' IS
*C LESS THAN ZERO, TO '1.4' IF IT IS EQUAL TO ZERO, AND TO '1.5' IF
*C IT IS GREATER THAN ZERO.  AT THOSE LINES, THE 'TYPE' COMMAND OUTPUTS
*C AN APPROPRIATE MESSAGE, THEN THE 'QUIT' COMMAND IS USED TO
*C STOP THE EXECUTION ENTIRELY.  IF THE 'QUIT' COMMAND WERE NOT THERE,
*C FOCAL WOULD HAVE CONTINUED EXECUTION WITH THE NEXT LINE IN
*C SEQUENCE.  THE 'QUIT' COMMAND MAY BE USED AT ANY PLACE IN A FOCAL
*C PROGRAM TO CAUSE THE EXECUTION OF STORED INSTRUCTIONS TO STOP
*C ENTIRELY, AND FOCAL TO PROMPT WITH A '*', AND AWAIT A NEW COMMAND.
*C MORE EXAMPLES:
*
*
*M 1.2

 1.20 IF (X-7)1.3,1.4,1.5
*M 1.3

 1.30 T "THE NUMBER IS LESS THAN ZERO\\\\SEVEN",!!!QUIT
*M 1.4

 1.40 T "THE NUMBER IS EQUAL TO ZERO\\\\SEVEN",!!!QUIT
*M 1.5

 1.50 T "THE NUMBER IS GREATER THAN ZERO\\\\SEVEN",!!!QUIT
*
```

```
*
*W

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

 1.10 A !,"ENTER A NUMBER:",X,!
 1.20 IF (X-7)1.3,1.4,1.5
 1.30 T "THE NUMBER IS LESS THAN SEVEN",!!;QUIT
 1.40 T "THE NUMBER IS EQUAL TO SEVEN",!!;QUIT
 1.50 T "THE NUMBER IS GREATER THAN SEVEN",!!;QUIT

*GO

ENTER A NUMBER:6

THE NUMBER IS LESS THAN SEVEN

*G

ENTER A NUMBER:9

THE NUMBER IS GREATER THAN SEVEN

*GO

ENTER A NUMBER:7

THE NUMBER IS EQUAL TO SEVEN

*G

ENTER A NUMBER:5+4

THE NUMBER IS GREATER THAN SEVEN

*GO

ENTER A NUMBER:9-12

THE NUMBER IS LESS THAN SEVEN

*
```

```
*
*C      IT IS NOT ALWAYS NECESSARY TO SUPPLY ALL THREE LINE NUMBERS WHEN
*C USING THE 'IF' COMMAND.  IF A LINE NUMBER IS OMITTED OR NULL (A COMMA
*C IS THERE, BUT NOTHING IS BEFORE IT), THEN FOCAL WILL PROCEED TO THE
*C NEXT COMMAND IN SEQUENCE (INSTEAD OF TRANSFERING CONTROL TO
*C A NEW PLACE) IF THE CONDITION IS TRUE (EXPRESSION LESS THAN
*C ZERO, EQUAL TO ZERO, OR GREATER THAN ZERO).  THIS ALLOWS FOR MORE
*C COMPACTNESS IN THE PROGRAM.  SOME EXAMPLES SHOULD HELP CLARIFY THIS!
*
*E ALL
*
*1.1 A !"ENTER A NUMBER:",X,!;I (X-7)1.5,1.6;T "I HAVE CONTINUED ON 1.1",!!Q
*1.5 T "I AM AT LINE 1.5",!
*1.6 T "I AM AT LINE 1.6",
*W

 C FOCAL-65 (V3D) 18-JUL-7

 1.10 A !"ENTER A NUMBER:",X,!;I (X-7)1.5,1.6;T "I HAVE CONTINUED ON 1.1",!!C
 1.50 T "I AM AT LINE 1.5",!!Q
 1.60 T "I AM AT LINE 1.6",!!Q

*GO

ENTER A NUMBER:3

I AM AT LINE 1.5

*
*G

ENTER A NUMBER:7

I AM AT LINE 1.6

*
*G

ENTER A NUMBER:9

I HAVE CONTINUED ON 1.1

*
```

```
*
*C   SINCE THE THIRD LINE NUMBER WAS OMITTED, FOCAL CONTINUED WITH THE NEXT
*C COMMAND IN SEQUENCE (THE 'T "I HAVE 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:
*
*M 1.1

 1.10 A !"ENTER A NUMBER:",X,!;I (X-7)1.5,1.6;\
*
*1.2 T "I AM AT LINE 1.2",!;Q
*
*W

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

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

*GO

ENTER A NUMBER:3

I AM AT LINE 1.5
*GO

ENTER A NUMBER:7

I AM AT LINE 1.6
*GO

ENTER A NUMBER:9

I 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:
*
*
*M 1.1

 1.10 A !"ENTER A NUMBER:",X,!;I (X-7)1.5
*
*
```

```
*
*W

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

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

*GO

ENTER A NUMBER:3

I AM AT LINE 1.5
*GO

ENTER A NUMBER:7

I AM AT LINE 1.2
*GO

ENTER A NUMBER:9

I AM AT LINE 1.2
*
*
*C   SINCE ONLY ONE LINE NUMBER WAS SPECIFIED (THE ONE TO TRANSFER TO
*C WHEN THE EXPRESSION WAS LESS THAN ZERO), FOCAL PROCEEDED TO THE NEXT COMMA
*C IN SEQUENCE WHEN THE VALUE OF THE EXPRESSION WAS EQUAL TO, OR GREATER THAN
*C ZERO.  MORE EXAMPLES:
*
*M 1.1

 1.10 A !"ENTER A NUMBER:",X,!!I (X-7)1.5\\\,1.6;T "I HAVE CONTINUED ON 1.1",
*W

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

 1.10 A !"ENTER A NUMBER:",X,!!I (X-7).1.6;T "I HAVE CONTINUED ON 1.1",!!Q
 1.20 T "I AM AT LINE 1.2",!!Q
 1.50 T "I AM AT LINE 1.5",!!Q
 1.60 T "I AM AT LINE 1.6",!!Q

*GO

ENTER A NUMBER:3

I HAVE CONTINUED ON 1.1
*G

ENTER A NUMBER:7

I AM AT LINE 1.6
*G

ENTER A NUMBER:9

I HAVE CONTINUED ON 1.1
*
```

```
*
*C   SINCE THERE WAS A COMMA, BUT NO LINE NUMBER, THEN FOCAL PROCEEDS TO
*C THE NEXT COMMAND IN SEQUENCE IF THE VALUE OF THE EXPRESSION IS LESS THAN
*C ZERO.  IF IT IS EQUAL TO ZERO, IT TRANSFERS TO LINE 1.6, IF IT IS GREATER
*C THAN ZERO, FOCAL PROCEEDS TO THE NEXT COMMAND IN SEQUENCE.  NOTE THIS!
*
*M 1.1

 1.10 A !"ENTER A NUMBER!",X,!!I (X-7),\1.6;T "I HAVE CONTINUED ON 1.1",!!Q
*W

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

 1.10 A !"ENTER A NUMBER!",X,!!I (X-7)1.6;T "I HAVE CONTINUED ON 1.1",!!Q
 1.20 T "I AM AT LINE 1.2",!!Q
 1.50 T "I AM AT LINE 1.5",!!Q
 1.60 T "I AM AT LINE 1.6",!!Q

*GO

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
*
```

```
*
*
*C    THAT LITTLE COMMA WAS VERY IMPORTANT!  IN THIS CASE FOCAL TRANSFERS
*C CONTROL TO LINE 1.6 IF THE VALUE OF THE EXPRESSION IS LESS THAN ZERO,
*C BUT PROCEEDS TO THE NEXT COMMAND IN SEQUENCE IF THE VALUE IS EQUAL TO,
*C OR GREATER THAN, ZERO (SAME AS ANOTHER EXAMPLE ABOVE).  ANOTHER EXAMPLE:
*
*M 1.1

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

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

 1.10 A !"ENTER A NUMBER:",X,!!I (X-7)1.6,,1.6;T "I HAVE CONTINUED ON 1.1",!
 1.20 T "I AM AT LINE 1.2",!!Q
 1.50 T "I AM AT LINE 1.5",!!Q
 1.60 T "I AM AT LINE 1.6",!!Q

*GO

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 AM AT LINE 1.6
*
```

```
*
*C   FOCAL WILL TRANSFER CONTROL TO LINE 1.6 IF THE VALUE OF THE EXPRESSION
*C IS NOT EQUAL TO ZERO (I.E. LESS THAN OR GREATER THAN), BUT WILL PROCEED
*C WITH THE NEXT COMMAND IN SEQUENCE IF THE VALUE IS EQUAL TO ZERO.  THE
*C 'IF' STATEMENT ALLOWS EITHER TWO OR THREE WAY BRANCHING OF PROGRAM CONTROL
*C DEPENDING UPON THE VALUE OF AN ARITHMETIC EXPRESSION.  THIS ALLOWS THE
*C COMPUTER PROGRAM TO COMPARE QUANTITIES AND PERFORM DIFFERENT COMMAND
*C SEQUENCES, DEPENDING UPON THE RELATIONSHIP OF THOSE QUANTITIES.  THUS,
*C THE 'IF' COMMAND IS A CONDITIONAL 'GOTO' COMMAND.
*
*C      IT IS DESIRABLE TO HAVE FOCAL REMEMBER WHERE IT IS EXECUTING COMMANDS ON
*C A GIVEN LINE, TRANSFER CONTROL TO, PERHAPS, ANOTHER LINE OR GROUP, THEN
*C HAVE FOCAL RETURN TO THE PLACE IT REMEMBERED IT WAS AT, CONTINUING TO
*C EXECUTE COMMANDS AS BEFORE.  THIS IS A VERY POWERFUL FEATURE, SINCE IT
*C ALLOWS THE PROGRAMMER TO WRITE A LINE (OR GROUP OF LINES) TO DO A
*C SPECIFIC TASK, AND WHENEVER THAT TASK NEEDS TO BE DONE, PERFORM THE LINE
*C OR GROUP, AND RETURN TO THE NEXT COMMAND IN SEQUENCE.  THIS CAPABILITY IS
*C PROVIDED FOR BY THE 'DO' COMMAND IN FOCAL.  THE 'DO' COMMAND CAN BE USED
*C TO PERFORM A SINGLE LINE, OR AN ENTIRE GROUP OF LINES.  WHEN FOCAL
*C ENCOUNTERS A 'DO' COMMAND, IT LOOKS FOR A LINE NUMBER OR A GROUP NUMBER
*C (SUCH AS '3') FOLLOWING THE 'DO' COMMAND.  FOCAL THEN REMEMBERS THE POSITION
*C TO RETURN TO AFTER PERFORMING THE LINE OR GROUP, TRANSFERS CONTROL TO THE
*C BEGINNING OF THE LINE OR GROUP, PERFORMS THE FOCAL COMMANDS THERE, THEN
*C RETURNS TO THE NEXT COMMAND IN SEQUENCE FOLLOWING THE 'DO' COMMAND.
*C THE 'DO' COMMAND OPERATES SLIGHTLY DIFFERENTLY WHEN PERFORMING ONLY A
*C SINGLE LINE OF COMMANDS, THAN IT DOES WHEN PERFORMING AN ENTIRE GROUP
*C OF COMMANDS.  LET US LOOK AT HOW THE 'DO' COMMAND FUNCTIONS WHEN WE
*C SPECIFY A SPECIFIC LINE NUMBER.  THE 'DO' CAUSES FOCAL TO REMEMBER WHERE
*C TO RETURN TO AFTER THE 'DO' HAS COMPLETED, TRANSFERS CONTROL TO THE
*C FIRST COMMAND ON THE SPECIFIED LINE, THEN PROCEEDS TO EXECUTE COMMANDS
*C AS THEY ARE ENCOUNTERED UNTIL A CARRIAGE RETURN IS ENCOUNTERED.  AT THE
*C TIME THE CARRIAGE RETURN IS ENCOUNTERED, CONTROL RETURNS TO THE PLACE
*C FOCAL REMEMBERED WHEN THE 'DO' WAS ENCOUNTERED.  SOME EXAMPLES:
*
*E ALL
*
*2.1 T "I AM EXECUTING COMMANDS ON LINE 2.1",!
*2.2 T "I AM EXECUTING COMMANDS ON LINE 2.2",!
*2.3 T "I AM EXECUTING COMMANDS ON LINE 2.3",!
*
*W

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

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

*
*C   NOTE THE ACTION OF THE 'GOTO' COMMAND:
*
*GO
I AM EXECUTING COMMANDS ON LINE 2.1
I AM EXECUTING COMMANDS ON LINE 2.2
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
*
```

```
*
*C    THE GOTO COMMAND DOES NOT REMEMBER ANY PLACE TO RETURN TO.
*C NOW NOTE THE ACTION OF THE 'DO' COMMAND:
*
*Do 2.1
I AM EXECUTING COMMANDS ON LINE 2.1
*
*Do 2.2
I AM EXECUTING COMMANDS ON LINE 2.2
*
*Do 2.3
I AM EXECUTING COMMANDS ON LINE 2.3
*
*D 2.2;T "THE 'DO' COMMAND RETURNED HERE",!;D 2.3;T "MORE COMMANDS ON THIS LIN
I AM EXECUTING COMMANDS ON LINE 2.2
THE 'DO' COMMAND RETURNED HERE
I AM EXECUTING COMMANDS ON LINE 2.3
MORE COMMANDS ON THIS LINE
*
*
*H 2.1

 2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1",!;G 2.3
*
*W

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

 2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1",!;G 2.3
 2.20 T "I AM EXECUTING COMMANDS ON LINE 2.2",!
 2.30 T "I AM EXECUTING COMMANDS ON LINE 2.3",!

*
*Do 2.1
I AM EXECUTING COMMANDS ON LINE 2.1
I AM EXECUTING COMMANDS ON LINE 2.3
*
```

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

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

 2.10 T "I AM EXECUTING COMMANDS ON LINE 2.1",!;G 2.3
 2.20 T "I AM EXECUTING COMMANDS ON LINE 2.2",!
 2.30 T "I AM EXECUTING COMMANDS ON LINE 2.3",!

*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
*
*D 2.3;D 2.2;T "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 !,"THE VALUE OF X= ",X,!
*W

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

 2.10 T !,"THE VALUE OF X= ",X,!

*S X=1;D 2.1;S X=2;D 2.1

THE VALUE OF X=      1.000

THE VALUE OF X=      2.000
*
*1.1 S X=X+1;D 2.1;G 1.1
*W

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

 1.10 S X=X+1;D 2.1;G 1.1

 2.10 T !,"THE VALUE OF X= ",X,!

*
*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

?-19 *  2.10    /
 T !,"THE VALUE OF X= ",X,!
     ^
*
```

```
*
*C   THE SEQUENCE OF COMMANDS IN LINE '1.1' ADDED ONE TO THE VALUE OF 'X',
*C PERFORMED THE COMMANDS ON LINE '2.1', THEN TRANSFERED BACK TO THE BEGINNING
*C OF LINE '1.1' AGAIN.  THIS SEQUENCE OF COMMANDS WOULD HAVE
*C CONTINUED TO DO THIS AD INFINITUM.  THIS IS CALLED AN 'INFINITE LOOP',
*C SINCE THERE IS NO WAY (NORMALLY) TO STOP EXECUTION.  IN THIS CASE,
*C I PRESSED THE 'INTERRUPT' BUTTON ON MY COMPUTER, WHICH CAUSED A
*C FOCAL ERROR, STOPPING THE EXECUTION, AND PRINTING THE APPROPRIATE
*C ERROR MESSAGES.  WHAT IF WE WANTED TO OUTPUT ONLY THE FIRST 10 VALUES
*C OF 'X', THEN CAUSE THE PROGRAM TO STOP?  EXAMPLE:
*
*M 1.1

 1.10 S X=X+1;D 2.1;I (X-10)1.1;T "THAT'S ALL!",!!;Q
*
*W

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

 1.10 S X=X+1;D 2.1;I (X-10)1.1;T "THAT'S ALL!",!!;Q

 2.10 T !,"THE VALUE OF X= ",X,!

*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
THAT'S ALL!

*
```

```
*
*C   THE 'IF' STATEMENT WAS USED TO COMPARE THE VALUE OF 'X' TO THE
*C NUMBER 10, AND AS LONG AS 'X' WAS LESS THAN 10 ( (X-10) LESS
*C THAN ZERO), THEN THE 'IF' COMMAND WOULD TRANSFER CONTROL BACK TO
*C THE BEGINNING OF LINE 1.1.  WHEN THE VALUE OF 'X' WAS EQUAL TO 10
*C ( (X-10) EQUAL TO ZERO), THEN THE NEXT COMMAND IN SEQUENCE WAS EXECUTED.
*C WHICH TYPED "THAT'S ALL!", AND QUIT.
*
*
*C    BACK TO THE 'DO' COMMAND.  IT DOES NOT MATTER WHEN OR WHERE THE
*C CARRIAGE RETURN IS ENCOUNTERED ONCE CONTROL HAS BEEN TRANSFERRED TO THE
*C SPECIFIC LINE TO 'DO', BUT WHEN IT DOES ENCOUNTER A CARRIAGE
*C RETURN, THEN CONTROL RETURNS TO THE NEXT COMMAND IN SEQUENCE AFTER THE
*C 'DO' WHICH TRANSFERRED CONTROL.  WHEN THE 'DO' COMMAND IS FOLLOWED BY
*C A GROUP NUMBER (SUCH AS '4'), THEN FOCAL REMEMBERS WHERE TO COME BACK TO
*C AND TRANSFERS CONTROL TO THE LOWEST NUMBERED STEP WITHIN THAT GROUP.  COMMAND
*C ARE EXECUTED AS THEY ARE ENCOUNTERED, UNTIL A CARRIAGE RETURN IS ENCOUNTERED
*C IN A LINE WHICH IS NOT PART OF THAT GROUP (I.E. IT HAS A DIFFERENT GROUP NUMBER
*C AT THAT POINT, CONTROL IS RETURNED TO THE PLACE REMEMBERED.  AT ANY TIME,
*C WHILE A 'DO' IS BEING PERFORMED, IMMEDIATE RETURN TO THE PLACE
*C REMEMBERED CAN BE FORCED BY USING THE 'RETURN' COMMAND.  SOME EXAMPLES:
*
*2.2 T !"THE VALUE OF X+2 IS = ",X+2,!
*W

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

 1.10 S X=X+1;D 2.1;I (X-10)1.1;T "THAT'S ALL!",!!;Q

 2.10 T !,"THE VALUE OF X= ",X,!
 2.20 T !"THE VALUE OF X+2 IS = ",X+2,!

*M 1.1

 1.10 S X=X+1;D 2.1\\;I (X-10)1.1;T "THAT'S ALL!",!!!Q
*W

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

 1.10 S X=X+1;D 2;I (X-10)1.1;T "THAT'S ALL!",!!;Q

 2.10 T !,"THE VALUE OF X= ",X,!
 2.20 T !"THE VALUE OF X+2 IS = ",X+2,!

*
```

```
*
#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↑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!

*
```

```
*
*M 1.1

 1.10 S X=X+1;D 2;I (X-10\\5)1.1;T "THAT'S ALL!",!!;Q
*2.15 I (X-3)2.2;R
*W

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

 1.10 S X=X+1;D 2;I (X-5)1.1;T "THAT'S ALL!",!!;Q

 2.10 T !,"THE VALUE OF X= ",X,!
 2.15 I (X-3)2.2;R
 2.20 T !"THE VALUE OF X+2 IS = ",X+2,!

*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 TRANSFERED CONTROL
*C TO LINE 2.2 (OUTPUTTING X+2) AS LONG AS X WAS LESS THAN 3.  IN
*C ALL OTHER CASES, AND IMMEDIATE 'RETURN' FROM GROUP 2 WAS EXECUTED.
*C MORE EXAMPLES:
*
*S X=20;D 2

THE VALUE OF X=    20.000
*
*S X=1;D 2

THE VALUE OF X=     1.000

THE VALUE OF X+2 IS =      1.000
*
*S X=20;D 2,2

THE VALUE OF X+2 IS =    400.000
*
```

```
*
*C   ANOTHER 'DO' COMMAND CAN BE ENCOUNTERED ANYTIME AFTER A 'DO'
*C COMMAND HAS TRANSFERED CONTROL TO A LINE OR GROUP.  FOCAL PROCESSES
*C THIS 'DO' COMMAND, BY REMEMBERING THERE TO COME BACK TO, TRANSFERING CONTROL
*C TO THE LINE OR GROUP, AND WHEN CONTROL RETURNS, IT WILL BE TO THE LAST
*C PLACE REMEMBERED.  THEN WHEN THE FIRST 'DO' IS OVER, CONTROL WILL RETURN
*C TO THE NEXT COMMAND IN SEQUENCE AFTER THE FIRST 'DO'.  THUS 'DO'
*C COMMANDS CAN BE NESTED.  THERE IS NO IMPLIED LIMIT ON THE DEPTH TO
*C WHICH FOCAL 'DO' COMMANDS MAY BE NESTED.  SOME EXAMPLES:
*
*W

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

 1.10 S X=X+1;D 2;I (X-5)1,1;T "THAT'S ALL!".!!!Q

 2.10 T !,"THE VALUE OF X= ",X,!
 2.15 I (X-3)2.2;R
 2.20 T !"THE VALUE OF X+2 IS = ",X+2,!

*M 2.15

 2.15 I (X-3)2,2;D 3;R
*3.1 T !"THE VALUE OF x+3 IS = ",X+3,!
*W

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

 1.10 S X=X+1;D 2;I (X-5)1,1;T "THAT'S ALL!",!!!Q

 2.10 T !,"THE VALUE OF X= ",X,!
 2.15 I (X-3)2.2;D 3;R
 2.20 T !"THE VALUE OF X+2 IS = ",X+2,!

 3.10 T !"THE VALUE OF X+3 IS = ",X+3,!

*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!

*
```

```
*
*C   NOW, IF THE VALUE OF X IS GREATER THAN, OR EQUAL TO 3, THE VALUE
*C OF X+3 IS OUTPUT INSTEAD OF THE VALUE OF X+2.  THE 'DO' COMMAND
*C IS A VERY USEFUL FACILITY, AND GREATLY INCREASES THE POWER
*C OF FOCAL.  THE 'IF' COMMAND PROVIDES A FACILITY TO PERFORM A 'GOTO'
*C BASED UPON THE VALUE OF AN ARITHMETIC EXPRESSION.  THE 'ON' COMMAND
*C PROVIDES THE ABILITY TO PERFORM A 'DO' COMMAND BASED ON THE VALUE
*C OF AN ARITHMETIC EXPRESSION.  IT WORKS IN THE SAME MANNER AS THE 'IF'
*C COMMAND, BUT INSTEAD OF TRANSFERING COMPLETELY TO THE SPECIFIED LINE
*C OR GROUP, A 'DO' COMMAND OF THE SPECIFIED LINE OR GROUP IS PERFORMED
*C AND, WHEN THE 'DO' COMES BACK, THE NEXT STATEMENT IN SEQUENCE IS
*C EXECUTED BY FOCAL, AS IN NORMAL SEQUENTIAL PROCESSING.  SOME EXAMPLES:
*
*
*E ALL
*W

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

*1.1 A !"NUMBER:",X,!;O (X-7)2.1,2.2,2.3;G 1.1
*
*2.1 T "I AM AT LINE 2.1",!
*2.2 T "I AM AT LINE 2.2",!
*2.3 T "I AM AT LINE 2.3".!
*
*W

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

 1.10 A !"NUMBER:",X,!;O (X-7)2.1,2.2,2.3;G 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",!

*G

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:
?-j9 @  1.10

 A !"NUMBER:",X,!;O (X-7)2.1,2.2,2.3;G 1.1


*
```

```
*
*C   THE 'INTERRUPT' BUTTON WAS PRESSED TO GET OUT OF THE ABOVE
*C INFINITE LOOP.  A 'DO' OF LINE 2.1 WAS PERFORMED IF THE VALUE OF
*C X WAS LESS THAN 7, A 'DO' OF LINE 2.2 WAS PERFORMED IF THE
*C VALUE WAS EQUAL TO 7, AND A 'DO' OF LINE 2.3 WAS PERFORMED IF THE
*C VALUE WAS GREATER THAN 7.  IN ALL CASES, CONTROL RETURNED TO THE
*C NEXT STATEMENT IN SEQUENCE WHICH FOLLOWED THE 'ON' COMMAND.  MORE
*C EXAMPLES!
*
*M 1.1

 1.10 A !"NUMBER:",X,!;O (X-7)2.1,2.2\\,2.3;G 1.1
*W

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

 1.10 A !"NUMBER:",X,!;O (X-7)2.1,2.2,2.3;G 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",!

*G

NUMBER:3

I AM AT LINE 2.1

NUMBER:7

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

NUMBER:9

I AM AT LINE 2.3

NUMBER:
?-19 @  1.10

 A !"NUMBER:",X,!;O (X-7)2.1,2.2,2.3;G 1.1


*
```

```
*
*C   IN THIS CASE, ALL OF GROUP 2 WAS PERFORMED WHEN THE VALUE OF X
*C WAS EQUAL TO SEVEN.  LINE AND/OR GROUP NUMBERS MAY BE OMITTED
*C (JUST AS IN THE 'IF' COMMAND), IN WHICH CASE CONTROL WILL SIMPLY
*C PASS TO THE NEXT COMMAND IN SEQUENCE (JUST AS IN THE 'IF' COMMAND).
*C SOME EXAMPLES:
*
*M 1.1

 1.10 A !"NUMBER:",X,!;O (X-7)2.1\\\,2,2.3;G 1.1
*
*W

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

 1.10 A !"NUMBER:",X,!;O (X-7),2,2.3;G 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",!

*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:3


NUMBER:2


NUMBER:8

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:
?-19 @  1.10

 A !"NUMBER:",X,!;O (X-7),2,2.3;G 1.1
                 ↑

*
```

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

 C FOCAL-65 (V3D) 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 $

X0( 0)=    2.000
*
*DO 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
*
```

```
*
*1.1 A !,"NUMBER:",X,!;O (X),Y;T "CONTINUING ON 1.1",!;G 1.1
*
*S Y=2.1
*
*G

NUMBER:-1

CONTINUING ON 1.1

NUMBER:1

CONTINUING ON 1.1

NUMBER:0

I AM AT LINE 2.1
CONTINUING ON 1.1

NUMBER:Y=2.2

CONTINUING ON 1.1

NUMBER:0

I AM AT LINE 2.2
CONTINUING ON 1.1

NUMBER:Y=2

CONTINUING ON 1.1

NUMBER:0

I AM AT LINE 2.1
I AM AT LINE 2.2
I AM AT LINE 2.3
CONTINUING ON 1.1

NUMBER:4

CONTINUING ON 1.1

NUMBER:-1

CONTINUING ON 1.1

NUMBER:
?-19 @  1.10

 A !,"NUMBER:",X,!;O (X),Y;T "CONTINUING ON 1.1",!;G 1.1


*
```

```
*
*C   THE 'Y=2' ABOVE SIMPLY SET THE VALUE OF 'Y' TO 2, INPUT THAT
*C VALUE IN THE 'ASK' COMMAND, ASSIGNED IT TO THE VARIABLE 'X', THE
*C VALUE OF THE EXPRESSION IN THE 'ON' COMMAND WAS THEN GREATER THAN
*C ZERO, SO FOCAL CONTINUED ON LINE 1.1.  HOWEVER, THE VALUE OF Y HAD
*C BEEN CHANGED TO 2, SO WHEN THE NEXT TIME A NUMBER WAS ASKED FOR, A
*C ZERO WAS ENTERED, 'X' WAS SET TO ZERO, AND THE 'ON' COMMAND
*C PERFORMED GROUP 'Y' (GROUP 2 IN THIS CASE), AND THEN RETURNED TO
*C CONTINUE PROCESSING THE NEXT SEQUENTIAL STATEMENT.
*C THE CAPABILITY TO ALLOW AN EXPRESSION TO DETERMINE THE VALUE OF
*C A LINE OR GROUP NUMBER EXPANDS THE POWER OF FOCAL.
*
*
*C     SOMETIMES A COMPUTER PROGRAM MUST REPEAT A PROCESS OVER AND OVER
*C SEVERAL TIMES (REMEMBER, COMPUTERS ARE GOOD AT THIS).  WE HAVE SEEN
*C ONE WAY TO DO THIS IN FOCAL ALREADY.  A VARIABLE CAN BE USED AS A
*C COUNTER AND GET INCREMENTED (OR DECREMENTED) EACH TIME THE PARTICULAR
*C PROCESS IS DONE.  AN 'IF' OR 'ON' STATEMENT CAN BE USED TO DETERMINE
*C IF THE PROCESS IS TO BE DONE AGAIN BY TESTING THE VALUE OF THE
*C COUNTING VARIABLE.  AN EXAMPLE FOLLOWS:
*
*E A
*W

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

*1.1 A !"NUMBER:",C,!
*
*1.1 A !"ENTER BEGINNING, INCREMENT, AND ENDING VALUES: ",B,I,E,!
*1.2 T "A PROCESS WITH B= ",B,!!! ((B=B+I)-E)1.2,1.1,2!Q
*
*W

 C 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,!!! ((B=B+I)-E)1.2,1.1,2!Q
```

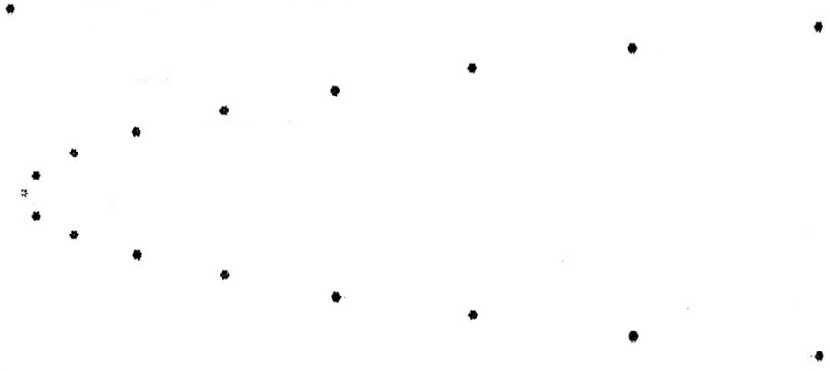
```
*
*GO

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
*
*GO

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
*
*GO

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
*
*
```

```
*
*C   THE ABOVE EXAMPLE ASKED FOR THREE VALUES, A BEGINNING ('B'),
*C AN INCREMENT ('I'), AND AND ENDING VALUE ('E').  THE COMMANDS AT LINE
*C 1.2 FORM A LOOP, WHERE THE !TYPE' COMMAND IS EXECUTED, THE
*C INCREMENT IS ADDED TO 'B' (AND BECOMES THE NEW VALUE OF 'B'), AND THEN 'B'
*C IS COMPARED TO 'E', THE ENDING VALUE.  IF 'B' IS LESS THAN, OR EQUAL TO, 'E'
*C THEN CONTROL GETS TRANSFERRED BACK TO LINE 1.2 AND THE 'TYPE' COMMAND IS
*C EXECUTED AGAIN.  IF THE VALUE OF 'B' IS GREATER THAN 'E', THEN CONTROL
*C PROCEEDS TO THE NEXT STATEMENT IN SEQUENCE, WHICH STOPS THE PROGRAM.
*C THIS TYPE OF LOOP IS OFTEN REQUIRED IN COMPUTER PROGRAMS, SO FOCAL
*C PROVIDES A MORE COMPACT METHOD FOR DOING A LOOP OF THIS TYPE.  THE
*C 'FOR' COMMAND ALLOWS THE PROGRAMMER TO PERFORM A LOOP IN THIS
*C MANNER.  LET'S FIRST LOOK AT THE ABOVE EXAMPLE, BUT WITH A 'FOR' LOOP
*C USED INSTEAD OF THE 'IF' LOOP.
*
*E A
*
*1.1 A !"BEGINNING, INCREMENT, ENDING VALUE: ",B,I,E;FOR X=B,I,E;T "A PROCESS
*                                                          WITH  X=",X,!
*W

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

 1.10 A !"BEGINNING, INCREMENT, ENDING VALUE: ",B,I,E;FOR X=B,I,E;T "A PROCESS
                                                          WITH X=",X,!
*GO

BEGINNING, INCREMENT, ENDING VALUE: 1,1,5
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=    3.000
A PROCESS WITH X=    5.000
A PROCESS WITH X=    7.000
*
*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
*
```

```
*
*C   THE 'FOR' COMMAND IS FOLLOWED BY A VARIABLE NAME WHICH IS USED
*C AS THE COUNTING VARIABLE.  UP TO THREE OPTIONS MAY BE SPECIFIED
*C FOLLOWING THE '=',  THESE ARE BEGINNING VALUE TO BE ASSIGNED TO
*C THE COUNTING VARIABLE, THE INCREMENT THAT IS TO BE ADDED ON TO THE
*C COUNTING VARAIBLE EACH TIME CONTROL IS RETURNED, AND THE ENDING VALUE
*C WHICH DETERMINES WHEN THE LOOPING PROCESS WILL NORMALLY TERMINATE.
*C THE EXACT OPERATION OF THE 'FOR' COMMAND IS AS FOLLOWS.  THE
*C COUNTING VARIABLE ('X' IN ABOVE EXAMPLE) IS SET EQUAL TO THE BEGINNING
*C VALUE ('B' IN EXAMPLE), THE INCREMENT AND THE ENDING VALUE ARE REMEMBERED
*C BY FOCAL, AS WELL AS THE START OF THE NEXT STATEMENT ON THE LINE.
*C A 'DO' OF ALL THE COMMANDS ON THE REST OF THE LINE IS PERFORMED.  THUS
*C CONTROL RETURNS WHEN A CARRIAGE RETURN IS ENCOUNTERED.  WHEN CONTROL
*C RETURNS, THE INCREMENT IS ADDED TO THE COUNTING VARIABLE (THE INCREMENT
*C MAY BE NEGATIVE, TO COUNT BACKWARDS), AND THEN THE COUNTING VARIABLE IS
*C COMPARED TO THE ENDING VALUE.  IF THE COUNTING VARIABLE IS LESS THAN.
*C OR EQUAL TO, THE ENDING VARIABLE (GREATER THAN OR EQUAL TO, IF THE
*C INCREMENT WAS NEGATIVE), THEN ANOTHER 'DO' OF THE REMAINDER OF THE LINE IS PER
*C OTHERWISE, THE 'FOR' LOOP HAS GONE TO COMPLETION, AND CONTROL IS       FORMED
*C TRANSFERRED TO THE BEGINNING OF THE NEXT FOCAL LINE NUMBER IN SEQUENCE.
*C NOTE THAT CONTROL IS TRANSFERRED TO THE NEXT LINE AND NOT TO THE NEXT
*C COMMAND, SINCE THE NEXT COMMAND, AND ALL THOSE ON THE REST OF THE LINE
*C WERE PART OF THE 'FOR' LOOP.  IF THE INCREMENT IS OMITTED (ONLY 2
*C PARAMETERS SPECIFIED), THEN A VALUE OF 1 IS ASSUMED.
*C SOME EXAMPLES:
*
*E A
*
*F X=1,2,10;T X,!
    1.000
    3.000
    5.000
    7.000
    9.000
*
*F X=1,10;T X,!
    1.000
    2.000
    3.000
    4.000
    5.000
    6.000
    7.000
    8.000
    9.000
   10.000
*
*
```

```
*
*F X=10,-1,1;T X,!
   10.000
    9.000
    8.000
    7.000
    6.000
    5.000
    4.000
    3.000
    2.000
    1.000
*
*S Y=7
*
*F X=1,Y;T X,!
    1.000
    2.000
    3.000
    4.000
    5.000
    6.000
    7.000
*
*F X=10,Y+10;T X,!
   10.000
   11.000
   12.000
   13.000
   14.000
   15.000
   16.000
   17.000
*
*C   SINCE A 'DO' OF THE REMAINDER OF THE LINE IS PERFORMED, AND SINCE
*C DO'S MAY BE NESTED, THEN THERE MAY BE A 'FOR' LOOP WITHIN A
*C 'FOR' LOOP.  SOME EXAMPLES!
*
*F X=1,5;T "X=",X,!!F Y=1,3;T "Y=",Y,!
X=    1.000
Y=    1.000
Y=    2.000
Y=    3.000
X=    2.000
Y=    1.000
Y=    2.000
Y=    3.000
X=    3.000
Y=    1.000
Y=    2.000
Y=    3.000
X=    4.000
Y=    1.000
Y=    2.000
Y=    3.000
X=    5.000
Y=    1.000
Y=    2.000
Y=    3.000
*
```

```
*
*C OR, WATCH THIS ONE:
*
*F X=-8.8;T "*"!;F Y=0.X+2;T " "
*
```



```
*
*
*C   A ONE LINE FOCAL PROGRAM PLOTS A PARABOLA:
*
```

```
*
*C   AS YOU HAVE SEEN, ANY OR ALL OF THE PARAMETERS IN THE 'FOR' COMMAND
*C CAN BE ARITHMETIC EXPRESSIONS (VERY POWERFUL).  IT MAY SEEM, AT FIRST
*C GLANCE, THAT THE FACT THAT THE 'FOR' COMMAND CAN ONLY 'DO' THE COMMANDS
*C ON THE REMAINDER OF THE LINE IS A SERIOUS LIMITATION.  NOT SO, BECAUSE
*C ONE OF THOSE COMMANDS CAN BE A 'DO' COMMAND TO PERFORM AS MANY
*C LINES OR GROUPS AS THE PROGRAMMER WANTS, AND CONTROL WILL RETURN
*C 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",!
*
*FOR 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
*
*F X=1,5;D 2.2;D 2.1
AT LINE 2.2
AT LINE 2.1
AT LINE 2.2
AT LINE 2.1
AT LINE 2.2
AT LINE 2.1
AT LINE 2.2
AT LINE 2.1
AT LINE 2.2
AT LINE 2.1
*
```

```
*
*C   A 'FOR' LOOP MAY BE TERMINATED AT ANY TIME IF A 'RETURN' COMMAND IS
*C ENCOUNTERED.  THIS RETURNS CONTROL BACK FROM THE 'DO' OF THE REMAINDER
*C OF THE COMMANDS AFTER THE 'FOR'.  THUS:
*
*E A
*
*1.1 F X=1,10;T "X=",X,!;I (X-6),1.3;
*1.2 T "THE FOR LOOP CAME HERE WHEN COMPLETED"!!;Q
*1.3 R
*
*W

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

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

*Go
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 $

X0( 0)=     6.000
*
*
*C   WHEN 'X' WAS EQUAL TO 6, CONTROL WAS TRANSFERED TO LINE 1.3,
*C WITHOUT HAVING SEEN A CARRIAGE RETURN, THE RETURN COMMAND RETURNED
*C CONTROL IMMEDIATELY BACK TO THE 'FOR' LOOP, BUT EXECUTION WAS
*C WAS STOPPED IN THE 'FOR' LOOP, AND CONTROL TRANSFERED TO THE NEXT
*C FOCAL LINE (IN THIS CASE 1.2).
*
```

```
*
*C     FOCAL ALLOWS FOR CERTAIN SPECIAL OPERATIONS TO BE PERFORMED WHICH
*C TRANSCEND NORMAL ARITHMETIC OPERATIONS.  THESE OPERATIONS ARE ACCOMPLISHED
*C THROUGH THE USE OF 'FUNCTIONS'.  A 'FUNCTION' IN FOCAL ALWAYS BEGINS
*C WITH THE LETTER 'F' (HENCE VARIABLE NAMES CANNOT BEGIN WITH THE LETTER)
*C AND HAVE A NAME (USUALLY 3 CHARACTERS) WHICH FOLLOWS THE 'F'.  A 'FUNCTION'
*C ALSO HAS AN 'ARGUMENT' LIST WHICH IS ENCLOSED IN PARENTHESES FOLLOWING
*C THE FUNCTION NAME.  SOME FUNCTIONS REQUIRE MORE ARGUMENTS THAN OTHERS,
*C DEPENDING UPON THE OPERATION PERFORMED BY THE FUNCTIONS.  IF A FUNCTION
*C REQUIRES ARGUMENTS, THEN THEY ARE PLACED INSIDE THE PARENTHESES AND
*C ARE SEPARATED WITH COMMAS.  A FUNCTION TAKES THE ARGUMENTS, PERFORMS
*C A SPECIFIC OPERATION USING THEM, AND ALL FOCAL FUNCTIONS RETURN AS THEIR
*C VALUE A SINGLE NUMBER.  A FUNCTION MAY APPEAR ANYPLACE IN AN ARITHMETIC
*C EXPRESSION THAT A NUMBER COULD APPEAR.  THE ARGUMENTS TO A FUNCTION
*C MAY BE ANY ARITHMETIC EXPRESSION, INCLUDING OTHER FUNCTION VALUES.  LET'S
*C LOOK AT A FEW SIMPLE FOCAL FUNCTIONS FIRST:
*
*
*C     THE 'FABS' FUNCTION TAKES ONE NUMERIC ARGUMENT AND THE VALUE IT
*C RETURNS IS THE ABSOLUTE VALUE OF THE ARGUMENT.  EXAMPLES:
*
*
*T FABS(-3),!
    3.000
*
*T 2+FABS(-3),!
    5.000
*
*T FABS(8),!
    8.000
*
*T FABS(-2.5)*2,!
    5.000
*
```

```
*
*C      THE 'FINT' FUNCTION TAKES ONE NUMERIC ARGUMENT AND THE VALUE IT
*C RETURNS IS THE 'GREATEST INTEGER LESS THAN THE NUMBER'.  THUS, THE
*C VALUE RETURNED WILL HAVE NO FRACTIONAL PART.  THIS FUNCTION DOES NO
*C ROUNDING (SEE 'FINR' BELOW) ON THE ARGUMENT.  EXAMPLES:
*
*
*T FINT(3.75),!
    3.000
*
*T FINT(3.99),!
    3.000
*
*T FINT(-3.14),!
   -4.000
*
*T FINT(5/2),!
    2.000
*
*T FABS(1+FINT(-3.14)),!
    3.000
*
*
*C      THE 'FINR' FUNCTION TAKES ONE NUMERIC ARGUMENT, ROUNDS TO THE
*C NEAREST WHOLE NUMBER, AND THEN PERFORMS THE 'GREATEST INTEGER LESS THAN
*C THE NUMBER' OPERATION.  EXAMPLES:
*
*
*T FINR(3.75),!
    4.000
*T FINR(3.99),!
    4.000
*T FINT(-3.14),!
   -4.000
*
*T FINR(-3.14),!
   -4.000
*
*T FINR(-3.5),!
   -4.000
*
*T FINR(2.5),!
    3.000
*
*T FINR(5/2),!
    3.000
*
```

```
*
*C     IT IS SOMETIMES USEFUL (FOR GAMES, SIMULATION, ETC.) TO HAVE THE
*C CAPABILITY TO GENERATE PSUEDO-RANDOM NUMBERS.  HOWEVER, WHEN DEBUGGING
*C (THE PROCESS OF REMOVING MISTAKES) SUCH PROGRAMS, IT IS NICE TO GET THE
*C SAME SEQUENCE OF RANDOM NUMBERS EACH TIME WE RUN THE PROGRAM.  WHEN
*C THE PROGRAM IS THEN WORKING, WE WOULD LIKE TO BE ABLE TO GET A DIFFERENT
*C SET OF RANDOM NUMBERS EACH TIME THE PROGRAM IS EXECUTED.  THE 'FRAN'
*C FUNCTION TAKES A SINGLE NUMERIC ARGUMENT, AND ALWAYS RETURNS AS ITS VALUE
*C A PSUEDO-RANDOM FRACTION BETWEEN Ø AND 1.  IF THE VALUE OF
*C THE ARGUMENT IS GREATER THAN ZERO, THEN THE RANDOM NUMBER GENERATOR
*C ROUTINE IS INITIALIZED TO GIVE A FIXED SEQUENCE OF RANDOM
*C NUMBERS.  IF THE VALUE OF THE ARGUMENT IS LESS THAN ZERO, THEN THE
*C RANDOM NUMBER GENERATOR IS INITIALIZED TO GIVE A RANDOM
*C SEQUENCE (DIFFERENT EACH TIME) OF RANDOM NUMBERS.  IF THE VALUE OF THE
*C ARGUMENT IS EQUAL TO ZERO, THEN THE NEXT RANDOM NUMBER FROM THE SEQUENCE
*C IS RETURNED.  SOME EXAMPLES:
*
*S FRAN(1)
*
*C   THIS 'SET' COMMAND CALLED THE 'FRAN' FUNCTION WITH AN ARGUMENT
*C GREATER THAN ZERO, WHICH INITIALIZED THE RANDOM NUMBER GENERATOR TO A FIXED
*C (REPEATABLE) SEQUENCE.  HOWEVER, THE VALUE RETURNED BY THE FUNCTION WAS
*C IGNORED, SINCE NO VARIABLE NAME SUBSTITUTION WAS INDICATED.
*C MORE EXAMPLES:
*
*
*FOR I=1,10!T FRAN(Ø),!
    Ø.996
    Ø.408
    Ø.497
    Ø.647
    Ø.240
    Ø.658
    Ø.284
    Ø.503
    Ø.695
    Ø.244
*
*FOR I=1,10!T FRAN(),!
    Ø.533
    Ø.944
    Ø.861
    Ø.667
    Ø.743
    Ø.473
    Ø.239
    Ø.426
    Ø.414
    Ø.588
*
```

```
*
*FOR I=1,10;T FRAN(),!
     0.329
     0.021
     0.047
     0.789
     0.630
     0.953
     0.141
     0.576
     0.642
     0.721
*
*S FRAN(1)
*
*F I=1,10;T FRAN(),!
     0.996
     0.408
     0.497
     0.647
     0.240
     0.658
     0.284
     0.503
     0.695
     0.244
*
*F I=1,10;T FRAN(),!
     0.533
     0.944
     0.861
     0.667
     0.743
     0.473
     0.239
     0.426
     0.414
     0.588
*
*F I=1,10;T FRAN(),!
     0.329
     0.021
     0.047
     0.789
     0.630
     0.953
     0.141
     0.576
     0.642
     0.721
*
```

```
*
*C   NOTICE THAT THE SAME SEQUENCE OF RANDOM VALUES WAS OBTAINED AFTER
*C AN 'FRAN(1)' CALL WAS ISSUED.  MORE EXAMPLES:
*
*S FRAN(1)
*
*S FRAN(-1)
*
*F I=1,10;T FRAN(),1
    0.057
    0.531
    0.495
    0.331
    0.423
    0.153
    0.904
    0.058
    1.000
    0.111
*
*
*C   A DIFFERENT SET OF RANDOM VALUES WAS OBTAINED AFTER CALLING
*C 'FRAN' WITH A NEGATIVE ARGUMENT.  RANDOM NUMBERS WITHIN ANY RANGE MAY BE
*C OBTAINED BY MULTIPLYING AND/OR ADDING APPROPRIATE SCALING FACTORS TO THE
*C FRACTION RETURNED BY 'FRAN'.  FOR EXAMPLE, IN ORDER TO OBTAIN RANDOM
*C INTEGERS WITHIN THE RANGE OF 0-9:
*
*
*F I=1,20;T FINT(FRAN()*10),1
    7.000
    3.000
    2.000
    9.000
    6.000
    7.000
    7.000
    0.000
    8.000
    7.000
    0.000
    0.000
    1.000
    8.000
    4.000
    3.000
    8.000
    4.000
    5.000
    4.000
*
*
```

```
*
*C     SOMETIMES WE NEED TO BE ABLE TO OUTPUT ANY CHARACTER WE WANT
*C TO AN OUTPUT DEVICE, OR BE ABLE TO INPUT ANY CHARACTER WE WANT TO FROM
*C AN INPUT DEVICE.  FOCAL HAS TWO SPECIAL FUNCTIONS FOR THIS PURPOSE.  THERE
*C IS A THING CALLED ASCII (AMERICAN STANDARD CODE FOR INFORMATION
*C INTERCHANGE) WHICH ASSIGNS A NUMERIC VALUE TO EACH OF THE POSSIBLE
*C CHARACTER  (THERE ARE 128 OF THEM).  FOR INSTANCE THE ASCII CODE FOR THE
*C CHARACTE   A' IS 65.  THE ASCII CODES CAN BE FOUND IN ANY
*C COMPUTER REFERENCE BOOK.  THE 'FOUT' FUNCTION TAKES ONE NUMERIC ARGUMENT
*C IN THE RANGE 0-255, AND OUTPUTS THE ASCII CHARACTER WHICH HAS THAT NUMBER
*C AS ITS CODE NUMBER.  THUS ANY CHARACTER CAN BE OUTPUT WITH AN 'FOUT' FUNCTION
*C IF THE PROGRAMMER SUPPLIES IT'S ASCII CODE AS THE ARGUMENT.  SOME EXAMPLES
*
*
*S FOUT(65)
A*
*
*S FOUT(66)!T !
B
*
*F I=0,25;S FOUT(65+I)
ABCDEFGHIJKLMNOPQRSTUVWXYZ*
*
*
*C   HAVE YOU EVER WONDERED HOW YOU WOULD TYPE OUT A '"' (DOUBLE QUOTE)?
*C HERE ARE SOME THINGS THAT DON'T WORK:
*
*T "
*
*T ""
*
*T """
*
*
*C   HERE IS SOMETHING THAT DOES WORK (KNOWING THAT THE ASCII CODE NUMBER
*C FOR A DOUBLE QUOTE IS 34)!
*
*
*S FOUT(34)
"*
*
*F I=1,10;S FOUT(34)
""""""""""*
*
```

```
*
*C   FOCAL ALLOWS US TO PRECEDE A SINGLE CHARACTER WITH A SINGLE QUOTE
*C MARK, AND THAT REPRESENTS A NUMBER WHOSE VALUE IS
*C THE ASCII CODE FOR THE SINGLE CHARACTER.  THUS, WE DON'T REALLY HAVE TO
*C KNOW WHAT THE CODE FOR A CHARACTER IS:
*
*T 'A,!
   65.000
*
*T 'B,!
   66.000
*
*T '",!
   34.000
*
*F I=0,25;S FOUT('A+I`
ABCDEFGHIJKLMNOPQRSTUVWXYZ*
*
*
*C     THE 'FCHR' FUNCTION REQUIRES NO ARGUMENT, INPUTS ONE CHARACTER
*C FROM THE INPUT DEVICE (ANY CHARACTER), AND THE NUMERIC VALUE RETURNED
*C AS THE VALUE OF THE FUNCTION IS THE ASCII CODE NUMBER FOR THAT CHARACTER.
*C FOR EXAMPLE:
*
*
*S Z=FCHR()
A*
*
*C   THE USER TYPED THE CHARACTER 'A' FROM THE KEYBOARD.
*
*T Z,!
   65.000
*
*
*E A
*
*F I=1,10;S C(I)=FCHR()
HI THERE!
*
```

```
*
*C   THE USER TYPED THE CHARACTERS 'HI THERE!' FOLLOWED BY A CARRIAGE
*C RETURN (THE 10TH CHARACTER).  THE ASCII CODES FOR THESE CHARACTERS
*C WERE STORED IN THE 'C' ARRAY.  TO SEE THEM:
*
*T S

I0( 0)=   11.000
C0( 1)=   72.000
C0( 2)=   73.000
C0( 3)=   32.000
C0( 4)=   84.000
C0( 5)=   72.000
C0( 6)=   69.000
C0( 7)=   82.000
C0( 8)=   69.000
C0( 9)=   33.000
C0(10)=   13.000
*
*
*C   TO WRITE THE CHARACTERS BACK OUT:
*
*F I=1,10;S FOUT(C(I))
HI THERE!
*
*
*C   TO SEE ONLY THE FIRST 5 CHARACTERS:
*
*F I=1,5;S FOUT(C(I))
HI TH*
*
*
*
*C   FOCAL HAS MORE POWERFUL FACILITITES FOR THE MANIPULATION OF CHARACTERS
*C WHICH IS EXPLAINED IN DETAIL LATER.  THE 'FOUT' AND 'FCHR' FUNCTIONS
*C ALLOW THE PROGRAMMER TO GET BY THOSE SEEMINGLY IMPOSSIBLE QUESTIONS
*C SUCH AS "HOW CAN I INPUT/OUTPUT THIS STRANGE CHARACTER?".
*
```

```
*
*C     FOCAL HAS THE ABILITY TO TRANSFER INFORMATION TO, AND OBTAIN INFOR-
*C MATION FROM, VARIOUS DEVICES THAT THE USER MAY HAVE ATTACHED TO HIS
*C COMPUTER SYSTEM.  THE VARIOUS HARDWARE DEVICES ARE ASSIGNED NUMBERS BY
*C THE FOCAL SYSTEM.  THESE ARE POSITIVE NUMBERS IN THE RANGE OF 0-127.
*C THE PROGRAMMER MAY INDEPENDENTLY CHANGE WHICH DEVICE FOCAL
*C IS INPUTTING FROM OR OUTPUTTING TO AT ANY GIVEN INSTANT, THROUGH THE
*C USE OF THE 'FIDV' AND 'FODV' FUNCTIONS.  THE 'FIDV' FUNCTION STORES
*C AWAY THE CURRENT DEVICE NUMBER OF THE CURRENT INPUT DEVICE, THEN TAKES
*C THE SINGLE NUMERIC ARGUMENT AS THE DEVICE NUMBER OF THE DEVICE TO MAKE
*C THE CURRENTLY ACTIVE INPUT DEVICE.  THE 'FODV' FUNCTION STORES AWAY
*C THE CURRENT DEVICE NUMBER OF THE CURRENT OUTPUT DEVICE, THEN TAKES THE
*C SINGLE NUMERIC ARGUMENT AS THE DEVICE NUMBER OF THE DEVICE TO MAKE THE
*C CURRENTLY ACTIVE OUTPUT DEVICE.  FURTHER INPUT/OUTPUT WILL TAKE
*C PLACE USING THE NEW DEVICES UNTIL EITHER A DIFFERENT DEVICE
*C IS MADE CURRENT THROUGH A NEW CALL TO 'FIDV'/'FODV' OR A 'RESTORE INPUT'
*C (ABBREVIATED 'R I') OR 'RESTORE OUTPUT' (ABBREVIATED 'R O') COMMAND IS
*C EXECUTED, WHICH RESTORES THE INPUT/OUTPUT DEVICE BACK TO WHAT IT WAS
*C JUST PRIOR TO THE LAST 'FIDV'/'FODV'.  IN THE FOLLOWING EXAMPLES, ASSUME
*C THAT THE CURRENT INPUT DEVICE IS DEVICE NUMBER 3, AND THE CURRENT OUTPUT
*C DEVICE IS DEVICE NUMBER 3.
*
*
*C   IN ORDER TO WRITE MY FOCAL PROGRAM TO OUTPUT DEVICE 0:
*
*
*S FODV(0);W;R O

 C FOCAL-69 (V3D) 18-JUL-77

*
*
*C   IN ORDER TO OUTPUT SOME NUMBERS TO OUTPUT DEVICE 0:
*
*
*F I=1,10;S FODV(3);T I,!;R O
    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 FODV(Ø);T I,!;R Ø
    1.ØØØ
    2.ØØØ
    3.ØØØ
    4.ØØØ
    5.ØØØ
    6.ØØØ
    7.ØØØ
    8.ØØØ
    9.ØØØ
   1Ø.ØØØ
*
*
*C TO INPUT 1Ø CHARACTERS FROM INPUT DEVICE Ø:
*
*
*1.1 S FIDV(Ø)
*1.2 F I=1,1Ø;S C(I)=FCHR()
*1.3 R I
*
*W

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

 1.1Ø S FIDV(Ø)
 1.2Ø F I=1,1Ø;S C(I)=FCHR()
 1.3Ø R I

*G
ABCDEFGHIJ*
*
*T $

IØ( Ø)=   11.ØØØ
CØ( 1)=   65.ØØØ
CØ( 2)=   66.ØØØ
CØ( 3)=   67.ØØØ
CØ( 4)=   68.ØØØ
CØ( 5)=   69.ØØØ
CØ( 6)=   7Ø.ØØØ
CØ( 7)=   71.ØØØ
CØ( 8)=   72.ØØØ
CØ( 9)=   73.ØØØ
CØ(1Ø)=   74.ØØØ
*
*F I=1,1Ø;S FOUT(C(I))
ABCDEFGHIJ*
*
*ERASE ALL
*
```

```
*
*C     SOMETIMES IT IS NECCESSARY TO 'INITIALIZE' A DEVICE BEFORE IT
*C CAN BE USED TO TRANSFER DATA.  SOME DEVICES REQUIRE IT, OTHERS DON'T.
*C FOR EXAMPLE, CASSETTES DO REQUIRE INITIALIZATION TO ALLOCATE BUFFER
*C SPACE FOR DATA STORAGE, ETC., BUT TELETYPES MAY NOT REQUIRE ANY
*C INITIALIZATION IN ORDER TO BE USED.  (THE TELETYPE INTERFACE MIGHT,
*C HOWEVER).  ANYWAY, IT IS GOOD FOCAL PROGRAMMING PRACTICE
*C TO 'INITIALIZE' ALL DEVICES BEFORE DATA TRANSFER TAKES PLACE.  THIS IS
*C ACCOMPLISHED USING THE 'FINI' AND THE 'FINO' FUNCTIONS.  'FINI' CALLS
*C A DEVICE DEPENDENT ROUTINE WITHIN FOCAL TO INITIALIZE A GIVEN DEVICE
*C FOR INPUT.  'FINO' CALLS A DEVICE DEPENDENT ROUTINE WITHIN FOCAL TO
*C INITIALIZE A GIVEN DEVICE FOR OUTPUT.  THE DEVICE NUMBER OF THE DEVICE
*C IS THE VALUE OF THE SINGLE ARGUMENT.  SOME DEVICES MUST ALSO BE 'CLOSED',
*C OR TOLD THAT INPUT/OUTPUT IS OVER, SO THAT THEY
*C CAN FINISH ANY INCOMPLETED TRANSFERS (BUFFERED CASSETTE I-O IS ONE EXAMPLE
*C THE DEVICE CAN BE 'CLOSED' BY THE USE OF THE 'FCLI' FUNCTION IF THE
*C DEVICE WAS INITIALIZED FOR INPUT, OR THE 'FCLO' FUNCTION IF THE DEVICE
*C WAS INITIALIZED FOR OUTPUT.  THESE FUNCTIONS ACCEPT A SINGLE
*C NUMERIC ARGUMENT WHICH IS THE DEVICE NUMBER OF THE DEVICE TO 'CLOSE'.
*C IT IS GOOD FOCAL PROGRAMMING PRACTICE TO 'CLOSE' A DEVICE
*C WHEN ALL INPUT/OUTPUT TO THAT DEVICE HAS BEEN COMPLETED.  SOME
*C EXAMPLES:
*
*
*E A
*
*1.1 S FINO(1),FODV(1);W!S FCLO(1);R 0
*
*W

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

 1.10 S FINO(1),FODV(1);W!S FCLO(1);R 0

*
*
*C   THE ABOVE SEQUENCE WILL INITIALIZE DEVICE NUMBER 1 FOR OUTPUT,
*C SET DEVICE NUMBER 1 AS THE CURRENT OUTPUT DEVICE, WRITE THE ENTIRE
*C FOCAL PROGRAM TO THE DEVICE, CLOSE THE DEVICE, AND RESTORE THE
*C OUTPUT DEVICE BACK TO WHATEVER IT WAS BEFORE THE 'FODV'.  IF
*C DEVICE 1 WAS A CASSETTE, THE ENTIRE FOCAL PROGRAM WOULD HAVE
*C BEEN STORED ON THE CASSETTE.
*
*ERASE ALL
*
```

```
*
*C   THIS ALLOWS THE USER TO STORE FOCAL PROGRAMS AND DATA ONTO OTHER
*C DEVICES THAT MAY BE CONNECTED TO HIS COMPUTER,  THERE IS ONE DEVICE
*C WHICH DOES HAVE SOME SPECIAL SIGNIFICANCE TO FOCAL.  THAT IS THE
*C USER'S CONSOLE DEVICE (THE DEVICE THAT HIS CONSOLE KEYBOARD AND
*C OUTPUT DEVICE, TELETYPE, CRT, ETC, IS CONNECTED TO),  ALL ERROR
*C MESSAGES ARE OUTPUT TO THE USER'S CONSOLE DEVICE.  THE USER MAY
*C CHANGE HIS CONSOLE DEVICE TO BE ANOTHER DEVICE ON THE COMPUTER
*C SYSTEM WITH THE 'FCON' FUNCTION.  THIS FUNCTION ACCEPTS A SINGLE
*C NUMERIC ARGUMENT AND THAT NUMBER (0-127) BECOMES THE NEW
*C CONSOLE DEVICE NUMBER,  THAT DEVICE STAYS THE CONSOLE DEVICE UNTIL
*C CHANGED BY THE USER WITH ANOTHER 'FCON' CALL.  A NEGATIVE ARGUMENT
*C TO 'FCON' DOES NOTHING, BUT RETURNS THE DEVICE NUMBER OF THE CURRENT
*C CONSOLE DEVICE.  SOME EXAMPLES:
*
*
*C   TO FIND OUT WHAT THE CURRENT CONSOLE DEVICE'S NUMBER IS:
*
*T FCON(-1),!
   3.000
*
*C   TO MOVE THE CONSOLE TO DEVICE 0!
*
*S FCON(0)
*
*C   THE CURRENT CONSOLE DEVICE IS DEVICE 0, BECAUSE!
*
*T FCON(-1),!
   0.000
*
*
*C   TO GO BACK TO DEVICE 3 AS THE CONSOLE!
*
*
*S FCON(3)
*
```

```
*
*C     FOCAL ALLOWS FOR THE PROGRAMMER TO MANIPULATE 'BYTE' OR
*C 'CHARACTER' STRINGS, AND PROVIDES SEVERAL FUNCTIONS WHICH FACILITATE
*C SUCH OPERATIONS.  LET'S LOOK AT 'BYTE' STRINGS FIRST.  UP TO THIS
*C POINT, ONLY NUMERICAL INFORMATION HAS BEEN READILY MANIPULATED.  EVEN
*C THE 'FOUT' AND 'FCHR' FUNCTIONS USED NUMBERS TO
*C REPRESENT THE CHARACTERS.  THE MAIN PROBLEMS ARE THAT THE 'FOUT' AND
*C 'FCHR' FUNCTIONS GIVE THE USER LITTLE FLEXIBILITY IN MANIPULATING
*C SERIES OF CHARACTERS OR 'BYTES', AND TO STORE A CHARACTER AS A
*C NUMBER IN A NUMERIC VARIABLE NAME TAKES ABOUT 7 TIMES THE AMOUNT OF
*C COMPUTER MEMORY STORAGE THAN MORE OPTIMAL METHODS.  THUS
*C FOCAL ALLOWS THE USER TO DEFINE AND USE 'BYTE' OR 'STRING' VARIABLES,
*C AS THEY ARE CALLED.  A 'STRING' VARIABLE IS A SEQUENTIAL SERIES
*C OF 'BYTES' STORED IN THE COMPUTER'S MEMORY.  IN GENERAL, NUMBERS
*C IN THE RANGE 0-255 MAY BE STORED IN EACH 'BYTE' POSITION.  IF THAT NUMBER
*C WERE TO BE AN ASCII CODE NUMBER, THEN A CHARACTER COULD ALSO BE
*C STORED THERE.  FOCAL DOES NOT CARE WHAT THE INFORMATION IS OR
*C WHAT IT REPRESENTS.  A 'BYTE' STRING IS JUST A SERIES OF NUMBERS IN
*C THE RANGE 0-255.  'BYTE' STRINGS ARE GIVEN VARIABLE NAMES, JUST LIKE
*C NUMERIC VARIABLE NAMES (A-Z FOR FIRST CHAR, BUT NOT F.  0-7 FOR SECOND CHAR.
*C SEE EARLIER DISCUSSION OF VARIABLE NAMES).  HOWEVER A '$' IS ADDED TO THE
*C NAME IN ORDER TO IDENTIFY THAT VARIABLE NAME AS REPRESENTING A STRING
*C OF BYTES.  THUS 'A' IS A NUMERIC VARIABLE AND 'A$' IS A BYTE VARIABLE.
*C THE SUBSCRIPT USED WITH THE BYTE VARIABLE DETERMINES WHICH BYTE
*C IN THE STRING IS BEING REFERENCED (THE FIRST BYTE IS SUBSCRIPTED 0, THE
*C SECOND 1, THE THIRD 2, AND SO ON).  A BYTE STRING MAY HAVE UP TO 250
*C BYTES STORED IN IT.  IF A BYTE STRING HAS NOT BEEN ASSIGNED A LENGTH,
*C (SEE 'FISL' FUNCTION), THEN THE DEFAULT LENGTH OF 72 BYTES IS ASSIGNED
*C TO IT THE FIRST TIME THAT IT IS REFERENCED.  ALSO ALL BYTES IN A STRING
*C ARE INITIALIZED TO THE ASCII CODE FOR A BLANK (32), WHEN FIRST
*C INITIALIZED.  HERE ARE SOME EXAMPLES WHICH ILLUSTRATE STRINGS IN THE
*C MOST PRIMITIVE MANIPULATIONS.  IF A STRING VARIABLE IS USED IN AN
*C ARITHMETIC EXPRESSION, THEN THE VALUE OF THAT VARIABLE IS THE NUMERIC
*C VALUE OF THE BYTE STORED THERE (0-255).  IF A CHARACTER IS STORED
*C THERE, THEN THE NUMBER WILL BE THE ASCII CODE NUMBER FOR THAT CHARACTER.
*
```

```
*
*C SOME EXAMPLES:
*
*S A$(0)=65
*
*T $

A0$="A                                                                    "
*
*
*C   REMEMBER THE ASCII CODE NUMBER 65 REPRESENTS THE CHARACTER 'A'.
*
*S A$(1)=66
*
*T $

A0$="AB                                                                   "
*
*S A$(2)='C
*T $

A0$="ABC                                                                  "
*
*T A$(1),!
   66.000
*
*T A$(2),!
   67.000
*
*S FOUT(A$(1))
B=
*
*F I=0,25;S A$(I)='A+I
*T $

A0$="ABCDEFGHIJKLMNOPQRSTUVWXYZ                                           "
I0( 0)=    26.000
*
```

```
*
*C   THE '$' OPTION OF THE 'TYPE' COMMAND OUTPUTS THE BYTE STRING
*C ASSUMING THAT ASCII CHARACTER CODE NUMBERS ARE STORED IN EACH BYTE
*C POSITION,  NOTE THAT 'A$' AND 'A$(Ø)' ARE THE SAME THING,
*C NAMELY THE FIRST BYTE IN THE STRING WHOSE NAME IS 'A' (OR 'AØ', SINCE
*C THEY ARE THE SAME.  AGAIN REFER TO EARLIER DISCUSSION OF VARIABLE NAMES).
*
*C   IF WE WANTED TO COPY THE CHARACTERS IN 'A$' INTO ANOTHER STRING,
*C SAY 'B$', A CRUDE WAY TO DO THAT MIGHT BE:
*
*
*F I=Ø,71;S B$(I)=A$(I)
*
*T $

AØ$="ABCDEFGHIJKLMNOPQRSTUVWXYZ                                              "
IØ( Ø)=    72,ØØØ
BØ$="ABCDEFGHIJKLMNOPQRSTUVWXYZ                                              "
*
*
*C   WE COULD INPUT 1Ø CHARACTERS INTO 'A$' BEGINNING AT SUBSCRIPT 3:
*
*
*F I=3,13;S A$(I)=FCHR()
HELLO,OVER!*
*
*T $

AØ$="ABCHELLO,OVER!OPQRSTUVWXYZ                                              "
IØ( Ø)=    14,ØØØ
BØ$="ABCDEFGHIJKLMNOPQRSTUVWXYZ                                              "
*
*
```

```
*
*C      THESE ARE VERY CRUDE MANIPULATIONS OF THE CHARACTERS.  LET'S NOW
*C LOOK AT SEVERAL FOCAL FUNCTIONS WHICH ALLOW MORE CONVENIENT MANIPULATION
*C OF CHARACTER STRINGS.  THE 'FISL' FUNCTION ALLOWS THE LENGTH OF
*C A CHARACTER STRING TO BE DEFINED, THE FIRST TIME IT IS REFERENCED.  IT
*C ALLOWS THE PROGRAMMER TO SET ASIDE IN THE COMPUTER'S MEMORY ONLY THE
*C THE NUMBER OF BYTES HE NEEDS (PROGRAMMER MAY NEED MORE, OR LESS, THAN THE
*C DEFAULT VALUE OF 72).  THE 'FISL' ACCEPTS AN ARBITRARY NUMBER OF ARGUMENTS
*C IN PAIRS.  THE FIRST ARGUMENT OF THE PAIR IS A NUMERIC VALUE WHICH IS THE
*C NUMBER OF BYTES TO SET ASIDE IN MEMORY FOR THE SPECIFIED STRING, AND THE
*C SECOND ARGUMENT OF THE PAIR IS THE NAME OF THE STRING VARIABLE.  ANY NUMBER
*C OF STRING VARIABLE'S LENGTHS MAY BE INITIALIZED IN ONE CALL TO 'FISL'
*C BY JUST PLACING MORE ARGUMENT PAIRS IN THE CALL.  IN ORDER FOR THE 'FISL'
*C TO WORK, THE CALL MUST BE THE FIRST TIME THAT THE SPECIFIED STRING
*C VARIABLE HAS BEEN REFERENCED IN THE FOCAL PROGRAM.  SOME EXAMPLES:
*
*
*ERASE ALL
*
*T $

*S FISL(16,A$,20,B$)
*
*T $

A0$="                "
B0$="                    "
*
*
*C   THIS CALL SET ASIDE 16 CHARACTERS FOR 'A$' AND 20 CHARACTERS FOR 'B$'.
*C ALL THE BYTES WERE SET TO CONTAIN THE ASCII CODE FOR A SPACE.
*C THE STRINGS MAY BE USED NOW!
*
*F I=0,15;S A$(I)='A+I
*T $

A0$="ABCDEFGHIJKLMNOP"
B0$="                    "
I0( 0)=   16.000
*
*
*C      SO IF THE PROGRAMMER ONLY NEEDS A STRING WITH 5 BYTES IN IT,
*C ONLY 5 BYTES NEED TO BE ALLOCATED.  ONCE A STRING HAS BEEN ALLOCATED, THEN
*C IT'S LENGTH MAY NOT BE CHANGED UNTIL THE VARIABLE LIST IS ERASED.
*
```

```
*
*C      IT IS USEFUL TO BE ABLE TO INPUT STRINGS FROM THE INPUT DEVICE,
*C WHATEVER KIND OF DEVICE IT MIGHT BE.  THE 'FSTI' FUNCTION ALLOWS THE
*C INPUTTING OF CHARACTERS FROM THE INPUT DEVICE AND THEIR STORING INTO A
*C STRING VARIABLE.  THE 'FSTI' FUNCTION HAS TWO MANDATORY ARGUMENTS, AND
*C AN OPTIONAL THIRD ARGUMENT.  THE FIRST ARGUMENT IS THE MAXIMUM
*C NUMBER OF CHARACTERS TO INPUT.  THE SECOND ARGUMENT IS THE
*C STRING NAME AND SUBSCRIPT POSITION TO START PLACING CHARACTERS
*C FROM THE INPUT DEVICE INTO THE STRING.  THE THIRD ARGUMENT, IF SUPPLIED,
*C IS AN ASCII CODE NUMBER FOR A SINGLE CHARACTER.  THIS CHARACTER
*C IS CALLED THE 'TERMINATION CHARACTER'.  IF THE TERMINATION CHARACTER IS
*C READ FROM THE INPUT DEVICE, INPUT STOPS AND THE FUNCTION RETURNS.  THE
*C TERMINATION CHARACTER IS NOT STORED INTO THE STRING, BUT THE VALUE
*C RETURNED BY THE 'FSTI' FUNCTION IS THE ACTUAL NUMBER OF CHARACTERS THAT
*C WERE TRANSFERED FROM THE INPUT DEVICE AND STORED INTO THE STRING.  IF
*C THE INPUT DEVICE IS THE CONSOLE DEVICE, THEN RUBOUT PROCESSING WILL TAKE
*C PLACE, ALLOWING THE THE 'RUBBING OUT' OF TYPING MISTAKES.
*C
*C SOME EXAMPLES:
*
*
*ERASE ALL
*
*1.1 E
*1.2 S FISL(20,A$,20,B$);S Z=FSTI(20,A$,^X)
*W

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

 1.10 E
 1.20 S FISL(20,A$,20,B$);S Z=FSTI(20,A$,^X)

*GO
HELLOX*
*
*T S

A0$="HELLO               "
B0$="                    "
Z0( 0)=    5.000
*
```

```
*
*C   THE TERMINATION CHARACTER IN THE 'FSTI' WAS AN 'X', WHICH MEANT THAT
*C CHARACTERS WOULD BE TRANSFERED FROM THE INPUT DEVICE (KEYBOARD IN THIS
*C CASE) INTO 'A$' BEGINNING AT SUBSCRIPT 0, UNTIL EITHER 20
*C CHARACTERS HAVE BEEN TRANSFERRED, OR UNTIL THE TERMINATION
*C CHARACTER 'X' HAS BEEN READ FROM THE INPUT DEVICE.  THE VALUE RETURNED
*C BY THE FUNCTION IS THE ACTUAL NUMBER OF CHARACTERS TRANSFERRED ( IN THIS
*C CASE 5).  THE TERMINATION CHARACTER 'X' IS NOT STORED IN THE STRING.
*C MORE EXAMPLES
*
*G
ABCDEFGHIJKLMNOPQRST*
*
*T $

A0$="ABCDEFGHIJKLMNOPQRST"
B0$="                    "
Z0( 0)=    20.000
*
*G
X*
*
*T $

A0$="                    "
B0$="                    "
Z0( 0)=     0.000
*
*
*C   IN THIS CASE NO CHARACTERS WERE TRANSFERRED, BECAUSE THE TERMINATION
*C CHARACTER WAS INPUT AS THE FIRST CHARACTER READ.
*
*
*G
ZZZX*
*
*T $

A0$="ZZZ                 "
B0$="                    "
Z0( 0)=     3.000
*
*
```

```
*
*M 1.2

 1.20 S FISL(20,A$,20,B$);S Z=FSTI(20,A$(2),'X)
*W

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

 1.10 E
 1.20 S FISL(20,A$,20,B$);S Z=FSTI(20,A$(2),'X)

*GO
BCDX*
*
*T $

A0$="  BCD                "
B0$="                     "
Z0( 0)=    3.000
*
*
*C   THE CHARACTERS WERE STORED IN 'A$' BEGINNING AT SUBSCRIPT POSITION
*C 2 (THE THIRD CHARACTER).  RUBOUT PROCESSING IS ALLOWED ONLY IF THE
*C CONSOLE IS THE INPUT DEVICE.  EXAMPLE:
*
*GO
NOW IS\\\\\\\THIS IS A TESTX*
*
*T $

A0$="  THIS IS A TEST     "
B0$="                     "
Z0( 0)=   14.000
*
*
```

```
*
*C      THE 'FSTO' FUNCTION ALLOWS STRINGS TO BE TRANSFERRED EFFICIENTLY
*C TO THE OUTPUT DEVICE.  THE ARGUMENTS TO THE 'FSTI' AND 'FSTO' ARE
*C IDENTICAL, EXCEPT THAT IN 'FSTO' CHARACTERS ARE READ FROM THE STRING
*C BEGINNING AT THE SPECIFIED SUBSCRIPT POSITION, TRANSFERRED TO THE OUTPUT
*C DEVICE, UNTIL EITHER THE MAXIMUM NUMBER HAVE BEEN OUTPUT, OR UNTIL THE
*C TERMINATION CHARACTER HAS BEEN READ FROM THE STRING.  THE TERMINATION
*C CHARACTER IS NOT SENT TO THE OUTPUT DEVICE.  THE VALUE RETURNED BY
*C 'FSTO' IS THE ACTUAL NUMBER OF CHARACTERS TRANSFERRED TO THE OUTPUT DEVICE.
*C
*C  NOTE!  IF THE FIRST ARGUMENT TO AN 'FSTI' OR 'FSTO' IS NULL
*C         (A COMMA, BUT NOTHING BEFORE IT), THEN THE MAXIMUM
*C         NUMBER IS INFINITE.  IN THIS CASE A TERMINATION CHARACTER IS
*C         ADVISABLE.
*C
*C SOME EXAMPLES:
*
*
*T $

A@$="  THIS IS A TEST     "
B@$="                     "
Z@( @)=   14,@@@
*
*$ Z=FSTO(4,A$(2))
THIS*
*
*
*C   FOUR CHARACTERS WERE TRANSFERRED FROM 'A$' BEGINNING AT SUBSCRIPT 2, TO
*C THE OUTPUT DEVICE.
*
*
*$ Z=FSTO(14,A$(2))
THIS IS A TEST*
*
*
*$ A$(16)=',
*T $

A@$="  THIS IS A TEST,    "
B@$="                     "
Z@( @)=   14.@@@
*
*$ Z=FSTO(,A$,',)
  THIS IS A TEST*
*
*T $

A@$="  THIS IS A TEST,    "
B@$="                     "
Z@( @)=   16.@@@
*
*C   THERE WAS NO MAXIMUM, SO CHARACTERS WERE TRANSFERED FROM 'A$' TO
*C THE OUTPUT DEVICE UNTIL THE TERMINATION CHARACTER ',' WAS READ FROM THE
*C STRING.  THE TERMINATION CHARACTER WAS NOT OUTPUT, AND THE VALUE RETURNED
*C 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
*
```

```
*
*C     SOMETIMES IT IS USEFUL TO LOOK FOR A CERTAIN GROUP OF CHARACTERS
*C TO SEE IF THEY ARE CONTAINED WITHIN A STRING OF CHARACTERS.  LET
*C US ASSUME WE HAVE THE FOLLOWING STRING VARIABLES:
*
*E A
*
*S FSTI(,A$,',)
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID,*
*
*S FSTI(,B$,',)
MEN,*
*
*T $

A0$="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID                    "
B0$="MEN                                                                    "
*
*
*C   THE CHARACTERS BETWEEN SUBSCRIPTS 0 AND 2 INCLUSIVE ('MEN') IN 'B$'
*C CERTAINLY ARE A SERIES OF CHARACTERS.  IF WE WANT TO HAVE FOCAL
*C SEARCH 'A$' (0-71), LOOKING FOR THE CHARACTERS 'MEN', WE WOULD USE THE
*C 'FSLK' FUNCTION.  EXAMPLE:
*
*S Z=FSLK(B$(0),B$(2),A$(0),A$(71))
*
*T $

A0$="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID                    "
B0$="MEN                                                                    "
Z0( 0)=    29,000
*
```

```
*
*C   THE VALUE RETURNED BY THE FUNCTION IS THE SUBSCRIPT IN 'A$' WHERE THE
*C CHARACTERS 'MEN' WERE FOUND.  THE 'FSLK' FUNCTION REQUIRES TWO PAIRS
*C OF ARGUMENTS.  THE FIRST PAIR DEFINES THE BEGINNING AND ENDING POINT OF
*C SOME CHARACTER STRING WHICH IS A SUBSET OF A STRING (IN THIS CASE
*C 'B$' FROM BYTE 0 THRU BYTE 2).  THE SECOND PAIR
*C DEFINES ANOTHER BEGINNING AND ENDING POINT OF THE CHARACTER STRING TO
*C SEARCH (IN THIS CASE 'A$' FROM BYTE 0 THRU BYTE 71).  THE SECOND STRING
*C IS SEARCHED, LOOKING FOR THE FIRST STRING TO BE FOUND SOMEWHERE WITHIN IT.
*C IF THE FIRST STRING IS FOUND, THEN THE VALUE RETURNED BY THE FUNCTION
*C IS THE SUBSCRIPT OF WHERE THE MATCH WAS ENCOUNTERED.  IF IT WAS NOT
*C FOUND, THEN THE VALUE -1 IS RETURNED AS THE VALUE OF THE FUNCTION.
*C THUS ONE CAN INPUT A STRING FROM THE KEYBOARD, CHECK TO SEE IF IT IS
*C ONE OF A KNOWN SERIES OF WORDS, AND PROCEED ACCORDINGLY.
*C MORE EXAMPLES:
*
*S Z=FSLK(B$,B$(2),A$(20),A$(40))
*T $

A0$="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
B0$="MEN
Z0( 0)=   29.000
*
*S Z=FSLK(B$,B$,A$,A$(71))
*
*T $

A0$="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
B0$="MEN
Z0( 0)=   13.000
*
*
*C   THE 'M' WAS LOCATED AT CHARACTER SUBSCRIPT 13 IN 'A$'.
*
*
*S Z=FSLK(B$,B$(2),A$(30),A$(40))
*T $

A0$="NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID
B0$="MEN
Z0( 0)=   -1.000
*
*
*C  THE CHARACTER STRING 'MEN' WAS NOT LOCATED WITHIN 'A$' SUBSCRIPTS
*C 30-40.
*
```

```
*C     WE WILL NOW LOOK AT A VERY POWERFUL FACILITY OF FOCAL, THE ABILITY
*C TO DO INPUT AND OUTPUT TO STRING VARIABLES AS IF THEY WERE HARDWARE
*C INPUT AND OUTPUT DEVICES.  WE MAY SET A STRING VARIABLE (BEGINNING AT
*C A CERTAIN SUBSCRIPT) TO BE OUR CURRENT INPUT DEVICE OR OUR CURRENT OUTPUT
*C DEVICE WITH THE 'FIDV' OR 'FODV' FUNCTIONS (DESCRIBED EARLIER).  WHEN THIS
*C IS DONE, INFORMATION WILL BE READ OR WRITTEN TO THE STRING
*C VARIABLE WITH ANY FOCAL COMMAND THAT DOES INPUT OR OUTPUT.  SOME EXAMPLES;
*
*
*E A
*
*S FODV(A$);T "THIS IS SOME INFORMATION";R O
*
*T $

A0$="THIS IS SOME INFORMATION                                                "
*
*
*S FODV(A$);T "MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW";R O
*T $

A0$="MARY HAD A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW                   "
*
*
*S FODV(A$(2));T "XXX";R O
*T $

A0$="MAXXXHAD 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);T "YYY";R O
*T $

A0$="MAXXXYYY A LITTLE LAMB, IT'S FLEECE WAS WHITE AS SNOW                   "
*
*
*S FODV(-1);T "THIS IS NEAT!";R O
*
*T $

A0$="MAXXXYYYTHIS IS NEAT!B, IT'S FLEECE WAS WHITE AS SNOW                   "
*
```

```
*
*S FODV(B$)!T %1,2+2!R O
*
*T $

A0$="MAXXXYYYTHIS IS NEAT!B, IT'S FLEECE WAS WHITE AS SNOW                "
B0$="4                                                                    "
*
*
*S FODV(B$(2))!T 2+3!R O
*T $

A0$="MAXXXYYYTHIS IS NEAT!B, IT'S FLEECE WAS WHITE AS SNOW                "
B0$="4 8                                                                  "
*
*
*C   IT IS NOT NECESSARY IN FOCAL TO HAVE SPECIAL FUNCTIONS TO CONVERT
*C NUMBERS TO CHARACTERS AND CHARACTERS TO NUMBERS!
*
*
*S B$(1)=',!S B$(3)=',
*T $

A0$="MAXXXYYYTHIS IS NEAT!B, IT'S FLEECE WAS WHITE AS SNOW                "
B0$="4,8,                                                                 "
*
*
*C   WATCH THIS!
*
*
*S FIDV(B$)!ASK X,Y!R I
*
*T $

A0$="MAXXXYYYTHIS IS NEAT!B, IT'S FLEECE WAS WHITE AS SNOW                "
B0$="4,8,                                                                 "
X0( 0)=4
Y0( 0)=8
*
```

```
*
*C   CONSIDER THIS:
*
*
*ERASE ALL
*
*S FODV(A$);T "SOME DATA IN A STRING";R 0
*
*T $

A@$="SOME DATA IN A STRING                                                    "
*
*
*C   TO COPY 'A$' INTO 'B$':
*
*S FIDV(A$),FSTI(72,B$);R 1
*
*T $

A@$="SOME DATA IN A STRING                                                    "
B@$="SOME DATA IN A STRING                                                    "
*
*
*C   THIS SEQUENCE SETS 'A$' AS THE CURRENT INPUT DEVICE, AND THEN THE
*C 'FSTI' INPUTS 72 CHARACTERS FROM THE CURRENT INPUT DEVICE AND STORES THEM
*C INTO 'B$', BEGINNING AT SUBSCRIPT @.
*
*C   CONSIDER THIS:
*
*S FODV(C$);T "T 1+X,!",!;R 0
*
*T $

A@$="SOME DATA IN A STRING                                                    "
B@$="SOME DATA IN A STRING                                                    "
C@$="T 1+X,!
                                                                             "
*
```

```
*
*C   NOTICE THAT SOME CHARACTERS 'T 1+X,!' HAVE BEEN PLACED IN 'C$' AND THAT
*C A CARRIAGE RETURN HAS ALSO BEEN PLACED THERE (RIGHT AFTER THE '!'),
*C NOTICE THAT THE CHARACTERS IN 'C$' FORM A VALID FOCAL COMMAND SEQUENCE.  THE
*C SEQENCE WOULD BE PERFECTLY VALID IF IT WERE TYPED IN BY THE
*C USER OR STORED WITH A LINE NUMBER PRECEEDING 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:
*
*Do C$
1
*
*T %5.03!S X=5
*
*Do C$
    6.000
*F X=0,9!D C$
    1.000
    2.000
    3.000
    4.000
    5.000
    6.000
    7.000
    8.000
    9.000
   10.000
*
*
```

```
*
*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 HAT!
*
*
*E A
*
*1.1 A !"NUMBER:"N;D 1.2;G 1.1
*1.2 S D(I*I+1)=N-2*N=FINT(N/2);O (-N)1.2;T D(I);S I=I-1
*T %1
*
*W

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

 1.10 A !"NUMBER:"N;D 1.2;G 1.1
 1.20 S D(I*I+1)=N-2*N=FINT(N/2);O (-N)1.2;T D(I);S I=I-1

*
*Go

NUMBER:5
101
NUMBER:4
100
NUMBER:10
1010
NUMBER:100
1100100
NUMBER:1024
10000000000
NUMBER:1023
1111111111
NUMBER:511
111111111
NUMBER:255
11111111
NUMBER:2
10
NUMBER:
?-19 @  1.10

 A !"NUMBER:"N;D 1.2;G 1.1
              ^

*
*
*C    IF YOU THINK THAT'S NEAT, REFER TO THE 'FSBR' USER DEFINED FUNCTION
*C FACILITY, OR THE 'SOFTWARE PRIORITY INTERRUPT SYSTEM' ('FPIC') DESCRIBED
*C LATER ON.
*
```

```
*
*C      SOMETIMES IT IS USEFUL FOR THE PROGRAMMER TO DEFINE HIS OWN
*C LINE OR GROUP AS A FUNCTION.  THEN WHENEVER HE WANTS THAT
*C PARTICULAR FUNCTION INVOKED, HE USES THE 'FSBR' FUNCTION
*C TO INVOKE THE FOCAL COMMAND LINE OR GROUP.  ONE NUMERIC ARGUMENT
*C CAN BE PASSED TO THE ROUTINE AND THE ROUTINE CAN RETURN A SINGLE NUMERIC
*C VALUE FOR THE VALUE OF THE 'FSBR' FUNCTION.  THE ARGUMENT IS PASSED IN
*C A PARAMETER INDEPENDENT FASHION (HEAVY COMPUTER SCIENCE JARGON).
*C THERE ARE ACTUALLY TWO ARGUMENTS TO THE 'FSBR' FUNCTION.  THE FIRST
*C IS A LINE NUMBER OR GROUP NUMBER OF THE LINE OR GROUP TO 'DO' AS
*C THE FUNCTION (YES, AN ARITHMETIC EXPRESSION CAN BE HERE).  THE SECOND
*C IS THE NUMERIC ARGUMENT TO BE PASSED TO THE FUNCTION.  THE
*C PRECISE SEQUENCE IS AS FOLLOWS.  THE CURRENT VALUE OF THE VARIABLE '&'
*C IS PUSHED ON THE STACK, THE VARIABLE '&' IS SET EQUAL TO THE NUMERIC
*C ARGUMENT PASSED TO THE FUNCTION (SECOND ARG OF 'FSBR'), A 'DO' IS PERFORMED
*C OF THE SPECIFIED LINE OR GROUP (THE FIRST ARG OF 'FSBR'), WHEN THE 'DO'
*C RETURNS, THE VALUE RETURNED BY THE FUNCTION IS THE CURRENT VALUE OF '&',
*C AND THE OLD VALUE OF '&' IS RESTORED FROM THE STACK.  VARIABLE NAMES CAN
*C BEGIN WITH THE CHARACTER '&', HENCE '&0'-'&7' MAY BE USED AS VALID
*C VARIABLE NAMES IN FOCAL PROGRAMS.  HOWEVER, BY CONVENTION, A FOCAL PROGRAMM
*C SHOULD ONLY USE '&' VARIABLES IN A USER-DEFINED FUNCTION IN ORDER TO
*C BE ABLE TO WRITE USER DEFINED FUNCTIONS WHICH ARE INDEPENDENT OF CALLING
*C ROUTINE.  SOME EXAMPLES:
*
*
*E A
*
*99.1 S &=&/2
*
*
*C   I HAVE MADE A VERY SIMPLE FUNCTION WHICH WILL TAKE THE ARGUMENT
*C AND DIVIDE IT BY TWO.  I NOW CALL IT VIA 'FSBR'!
*
*T FSBR(99,10),!
5
*
*T %5.03
*
*FOR I=1,10;T I,FSBR(99,I),!
    1.000    0.500
    2.000    1.000
    3.000    1.500
    4.000    2.000
    5.000    2.500
    6.000    3.000
    7.000    3.500
    8.000    4.000
    9.000    4.500
   10.000    5.000
*
```

```
*
*C   LET'S MAKE A FUNCTION WHICH CONVERTS DEGREES TO RADIANS:
*
*
*99.1 S &=&*3.1415926/180
*
*W

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

99.10 S &=&*3.1415926/180

*
*
*F X=0,45,360;T X,FSBR(99,X),!
   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=0,45,360;T %5,X,%5.05,FSBR(99,X),!
   0     0.00000
  45     0.78540
  90     1.57079
 135     2.35619
 180     3.14159
 225     3.92699
 270     4.71238
 315     5.49778
 360     6.28318
*
*T %5.03
*
```

```
*
*C   LET'S WRITE A USER DEFINED FUNCTION WHICH TAKES THE SQUARE
*C ROOT OF THE ARGUMENT GIVEN IT,  THIS FUNCTION USES THE COMMON
*C 'NEWTON-RAPHSON' ITERATION.
*
*
*E A
*
*99.1 S &1=&,&=2,&3=.000001
*99.2 I ((&=(&+&2)/2)*&3=FABS(&-&2=&1/&))99.2
*
*W

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

99.10 S &1=&,&=2,&3=.000001
99.20 I ((&=(&+&2)/2)*&3=FABS(&-&2=&1/&))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=1,10;T %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!.
*
```

```
*
*C      THE 'FSBR' FUNCTION IS RECURSIVE (HEAVY COMPUTER SCIENCE JARGON).
*C AS ARE MOST FOCAL FUNCTIONS,  THIS IMPLEMENTAION OF FOCAL DOES NOT
*C HAVE ANY INTRINSIC FUNCTIONS TO DO SUCH THINGS AS TRIGONOMETRIC
*C FUNCTIONS (SIN, COSINE, LOG, EXP, ARCTAN, ETC.);  HOWEVER, THESE
*C FUNCTIONS CAN BE MADE AVAILABLE BY SIMPLY WRITING ROUTINES IN FOCAL
*C TO PERFORM THE NECESSARY CALCULATIONS, THEN CALL THEM FROM THE
*C APPLICATION PROGRAM WITH 'FSBR' CALLS.  FOCAL ROUTINES TO DO ALL
*C THE COMMON TRIG FUNCTIONS (USING 'FSBR') ARE GIVEN IN AN APPENDIX.
*C ALSO A ROUTINE TO OUTPUT A NUMBER IN 'E' FORMAT (SCIENTIFIC NOTATION)
*C IS ALSO SHOWN THERE.  IN MANY CASES, THESE ROUTINES ARE SMALLER IN
*C SIZE (BUT SLOWER) THAN THE EQUIVALENT ROUTINES WRITTEN IN ASSEMBLY
*C LANGUAGE.  HERE IS AN EXAMPLE OF A RECURSIVE 'FSBR' FUNCTION TO CALCULATE
*C THE FACTORAL OF A NUMBER.  THE FACTORAL OF 'N' IS DEFINED MATHEMATICALLY
*C TO BE = N*(N-1)*(N-2)*(N-3)*...*3*2*1, I.E. THE PRODUCT OF ALL THE
*C INTEGERS UP THRU 'N'.  THE EXAMPLE:
*
*
*ERASE ALL
*
*99.1 I (1-&)99.2;R
*99.2 S &=&*FSBR(99,&-1)
*
*W

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

99.10 I (1-&)99.2;R
99.20 S &=&*FSBR(99,&-1)

*
*
*T FSBR(99,3),!
    6.000
*
*T FSBR(99,4),!
   24.000
*
*T FSBR(99,5),!
  120.000
*
*
*F X=7,-1,1;T "THE FACTORAL OF ",X," IS ",FSBR(99,X),!
THE FACTORAL OF     7.000 IS  5040.000
THE FACTORAL OF     6.000 IS   720.000
THE FACTORAL OF     5.000 IS   120.000
THE FACTORAL OF     4.000 IS    24.000
THE FACTORAL OF     3.000 IS     6.000
THE FACTORAL OF     2.000 IS     2.000
THE FACTORAL OF     1.000 IS     1.000
*
```

```
*
*C      FOCAL HAS A POWERFUL FACILITY AIMED AT THE EXPERIMENTER AND
*C REAL-TIME USER.  A FOCAL PROGRAM CAN BE INTERRUPTED BY SOME
*C EXTERNAL EVENT (A DOOR OPENING, A PHONE RINGING, A BURGLAR ENTERING)
*C AND A 'DO' OF A SPECIFIED FOCAL LINE OR GROUP PERFORMED, AND CONTROL
*C AUTOMATICALLY RETURNED TO THE INTERRUPTED ROUTINE.  FUTHERMORE,
*C THE VARIOUS INTERRUPTING DEVICES CAN BE ASSIGNED A PRIORITY, AND THE
*C HIGHEST PRIORITY EVENT WILL BE THE FIRST SERVICED, THE SECOND HIGHEST
*C PRIORITY WILL BE THE NEXT SERVICED, AND SO ON.  FOCAL DOES NOT
*C KNOW (OR CARE) WHAT CAUSED THE EVENT TO HAPPEN, BUT DEALS WITH EVENTS
*C AS BITS (BINARY DIGITS) THAT ARE SET IN AN 'EVENT BYTE' STORED IN THE
*C COMPUTERS MEMORY.  THE EVENTS CORRESPOND TO BIT POSITIONS
*C IN THIS BYTE FROM RIGHT TO LEFT (RIGHT IS EVENT 1, LEFT IS EVENT 8),
*C EVENT 8 IS THE HIGHEST PRIORITY, AND EVENT 1 IS THE LOWEST.
*C WHENEVER FOCAL PROMPTS WITH A '*' (HAS NOTHING TO DO), IT DISABLES
*C THE ABILITY TO BE INTERRUPTED.  THUS, THE FOCAL PROGRAMMER
*C MUST ENABLE FOCAL TO LOOK AT THE EVENT BITS AND INTERRUPT THE FOCAL
*C PROGRAM WHEN A GIVEN EVENT (OR GROUP OF EVENTS) HAPPENS.  SOMEONE
*C MUST HAVE ALSO WRITTEN AN ASSEMBLY LANGUAGE ROUTINE TO SET THE
*C APPROPRIATE BITS IN THE EVENT BYTE WHEN THE PARTICULAR EVENT ACTUALLY
*C HAPPENS.  THE PROGRAMMER USES THE 'FPIC' FUNCTION TO MANIPULATE THE
*C 'SOFTWARE PRIORITY INTERRUPT SYSTEM'.  THE 'FPIC' FUNCTION TAKES
*C AN ARBITRARY NUMBER OF ARGUMENTS, IN PAIRS, AND USES THEM IN CONTROLLING
*C THE SOFTWARE INTERRUPT SYSTEM.  THE FIRST ARGUMENT OF A PAIR IS A
*C SOFTWARE EVENT NUMBER (1-8).  THE SECOND ARGUMENT IS A FOCAL LINE OR
*C GROUP NUMBER (ARITHMETIC EXPRESSIONS ARE ALLOWED HERE) TO 'DO' WHEN
*C THE SPECIFIED EVENT BIT GETS SET.  THE 'FPIC' CALL ENABLES FOCAL TO
*C CHECK THE EVENT BIT EACH TIME A NEW FOCAL COMMAND IS RETRIEVED, AND IF
*C THE BIT IS SET, A 'DO' OF THE SPECIFIED LINE OR GROUP WILL BE PERFORMED.
*C WHEN THE 'DO' RETURNS, FOCAL WILL CONTINUE THE INTERRUPTED STATEMENT.
*C FOCAL ALSO CLEARS THE EVENT BIT BEFORE IT PERFORMS THE 'DO'.  IF
*C SEVERAL EVENTS (BITS) HAVE BEEN ENABLED WITH THE 'FPIC' FUNCTION,
*C THEN FOCAL WILL PERFORM THE 'DO' FOR THE HIGHEST PRIORITY ONE FIRST,
*C THEN WHEN THAT 'DO' RETURNS, THE NEXT HIGHEST PRIORITY, ETC.  THE
*C VALUE RETURNED BY THE 'FPIC' FUNCTION IS A NUMBER BETWEEN
*C 0 AND 255 WHICH WILL HAVE BITS SET FOR EACH EVENT THAT HAS BEEN ENABLED.
*C THUS, IF EVENTS 1 AND 3 HAVE BEEN ENABLED, A VALUE OF 5 WILL BE RETURNED.
*C IF THE FIRST ARGUMENT OF A PAIR IS 0, THEN THE SECOND ARGUMENT IS A
*C NUMBER 0-255 WHICH IS TO INDICATE WHICH EVENTS ARE TO BE ENABLED.  THIS
*C ALLOWS THE SOFTWARE INTERRUPT SYSTEM TO BE TEMPROARILY DISABLED
*C (BY 'FPIC(0,0)'), AND ENABLED AGAIN (IN THE EXAMPLE BY 'FPIC(0,5)').
*C IF THE FIRST ARGUMENT IS 0 AND THE SECOND IS NEGATIVE, THEN THE 'FPIC'
*C FUNCTION DOES NOTHING BUT RETURN THE ENABLE BYTE (AS IT ALWAYS DOES).
*C NOTE THAT ALL ACTIVE EVENT CHANNELS SHOULD BE ENABLED WITH AN 'FPIC'
*C CALL WHICH SPECIFIES THE LINE OR GROUP TO 'DO' FIRST, BEFORE THEY ARE
*C DISABLED/ENABLED LATER, SINCE IT IS NECESSARY FOR FOCAL TO
*C STORE THE LINE NUMBER OR GROUP NUMBER IN INTERNAL TABLES.
*
```

```
*
*C     WELL, AFTER THAT LONG-WINDED EXPLANATION, LET'S LOOK AT AN
*C EXAMPLE.  LET US SAY THAT A SWITCH CONNECTED TO A DOOR WILL GENERATE
*C AN INTERRUPT TO THE COMPUTER AND A ROUTINE WILL SET EVENT BIT 1.
*C ALSO, LET US SAY THAT A THERMOCOUPLE CIRCUIT CONNECTED TO THE
*C ROAST IN THE OVEN WILL GENERATE AN INTERRUPT TO THE COMPUTER ANC A
*C ROUTINE WILL SET EVENT BIT 2 WHEN THE TEMPERATURE IN THE ROAST
*C REACHES A CERTAIN VALUE.  ALSO, LET US SAY THAT A PUSHBUTTON WILL
*C GENERATE AN INTERRUPT AND SET BIT 7 WHEN THE BUTTON IS PUSHED.
*C HERE IS AN EXAMPLE FOCAL PROGRAM WHICH WILL ENABLE FOCAL TO SENSE
*C THESE CONDITIONS, INTERRUPT THE PROGRAM (WHICH IS AN INFINITE LOOP),
*C AND INFORM THE USER THAT THE EVENTS HAVE HAPPENED.
*
*
*E A
*
*1.1 E
*1.2 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!";D 99
*92.1 T !"THE ROAST HAS REACHED TEMPERATURE!";D 99
*97.1 T !"INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM."!!;Q
*99.1 T "  X= ",X,!
*
*W

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

 1.10 E
 1.20 S FPIC(1,91,2,92,7,97)
 1.30 S X=X+1;G 1.3

91.10 T !"SOMEONE IS AT THE DOOR!";D 99

92.10 T !"THE ROAST HAS REACHED TEMPERATURE!";D 99

97.10 T !"INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM."!!;Q

99.10 T "  X= ",X,!

*
*GO

SOMEONE IS AT THE DOOR!  X=  2097.000

THE ROAST HAS REACHED TEMPERATURE!  X=  3064.000

INTERRUPT ON LEVEL 7, I'M STOPPING THIS PROGRAM.

*
```

```
*
*C  THE PROGRAM ENABLED FOCAL TO 'DO' GROUP 91 IF EVENT 1 WAS SET,
*C 92 IF EVENT 2 WAS SET, AND 97 IF EVENT 7 WAS SET.  THE PROGRAM
*C THEN ENTERED AN INFINITE LOOP INCREMENTING THE VARIABLE 'X'.  WHEN
*C SOMEONE OPENED THE DOOR, GROUP 91 WAS PERFORMED.  WHEN THE ROAST
*C REACHED TEMPERATURE, GROUP 92 WAS PERFORMED.  WHEN THE USER PRESSED
*C THE PUSHBUTTON, GROUP 97 WAS PERFORMED, AND THE PROGRAM WAS STOPPED.
*C LET'S SEE WHAT HAPPENS WHEN THE DOOR IS OPENED AND THE ROAST HAS
*C REACHED TEMPERATURE AT THE SAME TIME.
*
*
*GO

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.

*
*
*C   NOTE THAT GROUP 92 WAS PERFORMED FIRST, SINCE THE ROAST IS
*C ASSOCIATED WITH A HIGHER EVENT NUMBER.  HOWEVER, AS SOON AS
*C GROUP 92 RETURNED, THE DOOR INTERRUPT (GROUP 91) WAS PERFORMED
*C IMMEDIATELY (AS EVIDENCED BY THE FACT THAT 'X' DID NOT GET INCREMENTED).
*C WHEN THE PUSHBUTTON WAS PRESSED, GROUP 97 WAS PERFORMED AND THE
*C PROGRAM WAS STOPPED.  THE POSSIBLE USES OF THIS FACILITY ARE ALMOST
*C UNLIMITED.
*
```

```
*
*C      WE WILL NOW LOOK AT A FEW REMAINING MISCELLANEOUS FUNCTIONS.
*C THE 'FECH' FUNCTION ALLOWS THE USER TO ENABLE/DISABLE THE AUTOMATIC
*C ECHOING OF CHARACTERS READ FROM THE INPUT DEVICE.  'FECH(0)' ENABLES
*C THE ECHOING, AND 'FECH(1)' DISABLES THE ECHOING.
*
*C      THERE IS A FOCAL FUNCTION WHICH IS SPECIFIC TO THE CONSOLE
*C DEVICE.  THIS FUNCTION IS IN FOCAL PRIMARILY BY POPULAR DEMAND,
*C SINCE IT IS USEFUL FOR GAMES, ETC.  THE 'FCUR' FUNCTION ALLOWS THE
*C PROGRAMMER TO POSITION THE CURSOR ON CRT TYPE TERMINALS (IF HE HAS A
*C CRT TYPE TERMINAL) TO A GIVEN ROW AND COLUMN ON THE SCREEN, WITHOUT
*C DISTURBING OTHER INFORMATION ON THE SCREEN.  THE 'FCUR' FUNCTION
*C TAKES TWO ARGUMENTS.  THE FIRST IS THE ROW NUMBER (0-N) TO MOVE TO.
*C THE SECOND IS THE COLUMN NUMBER (0-N) TO MOVE TO.  THE USER
*C MUST PATCH HIS CRT SPECIFIC ROUTINE INTO FOCAL  SEE THE APPENDIX TO
*C FIND OUT HOW TO DO THIS.
*
*
*C      FOR THOSE HACKERS THAT INSIST ON SUCH THINGS, THE 'FMEM' FUNCTION
*C ALLOWS THE PROGRAMMER TO READ AND/OR WRITE INFORMATION FROM/INTO
*C STORAGE LOCATIONS IN HIS COMPUTER'S MEMORY.  THE 'FMEM' FUNCTION
*C IS ALWAYS GIVEN TWO ARGUMENTS, THE FIRST IS THE PAGE NUMBER (0-255)
*C AND THE SECOND IS THE LOCATION (0-255) WITHIN THAT PAGE.  THESE
*C TWO VALUES FORM THE ADDRESS IN THE MEMORY.  THUS TO ACCESS
*C MEMORY ADDRESS 0120 (HEXADECIMAL), THE ARGUMENTS WOULD BE
*C 1 (FOR THE PAGE NUMBER) AND 32 (FOR THE LOCATION WITHIN THAT PAGE).
*C (20 HEX = 32 DECIMAL).  IF ONLY 2 ARGUMENTS ARE SUPPLIED, THE
*C VALUE RETURNED BY THE 'FMEM' FUNCTION IS THE DATA VALUE STORED IN THAT
*C MEMORY ADDRESS (A NUMBER 0-255).  IF ANOTHER ARGUMENT FOLLOWS, THEN
*C THE VALUE OF THAT ARGUMENT (0-255) IS DEPOSITED (WRITTEN INTO) THE
*C MEMORY ADDRESS, AND TH  ALUE RETURNED BY THE 'FMEM' FUNCTION IS THE
*C DATA VALUE THAT WAS STORE  RE BEFORE THE NEW VALUE WAS DEPOSITED.
*C SEVERAL DEPOSITS MAY BE MADE .  I ONE CALL TO 'FMEM', BY GIVING SEVERAL
*C TRIPLES (GROUPS OF THREE) IN ONE CALL.  EACH ONE SPECIFIES A PAGE NUMBER,
*C LOCATION IN THE PAGE, AND DATA TO BE STORED THERE.  THE VALUE RETURNED
*C WILL BE THE DATA BYTE THAT WAS IN THE LAST LOCATION BEFORE THE NEW
*C DATA BYTE WAS STORED THERE.  THIS IS USEFUL FOR THOSE TWEEKS OF MEMORY
*C WHERE SEVERAL THINGS MUST BE TWEEKED BEFORE CONTROL IS RETURNED TO
*C THE FOCAL PROGRAM.  (YES, IT IS POSSIBLE TO MAKE CHANGES TO THE FOCAL
*C SYSTEM WITH A FOCAL PROGRAM.  YOU SHOULD BE VERY CAREFUL, AND KNOW
*C WHAT YOU ARE DOING.)
*
*T FMEM(1,32),!
   92.000
*
*T FMEM(1,32,16),!
   92.000
*
*T FMEM(1,32),!
   16.000
*
```

APPENDIX A

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

```
*C                                  APPENDIX    B
*
*
*
*         TRIG FUNCTIONS IMPLEMENTED VIA 'FSBR' FUNCTIONS AS FOCAL ROUTINES.
*
*
*W

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

93.01 C COS!93! C SIN!93.3
93.10 I (&↑2=.01)93.2!S &=&/2;D 93!S &=2*&↑2-1!R
93.20 S &=1-&↑2/2+&↑4/24-&↑6/720!R
93.30 S &=1.57080-&!D 93

94.01 C ASIN!94 !C ACOS!94.3
94.10 I (&↑2=.01)94.2!S &=&/(FSBR(99,1+&)+FSBR(99,1-&))!D 94!S &=2*&!R
94.20 S &=&+&↑3/6+.075*&↑5+&↑7/22.4!R
94.30 D 94!S &=1.570796-&!R

95.01 C ATAN
95.10 I (&↑2=.01)95.2!S &=&/(1+FSBR(99,&↑2+1))!D 95!S &=2*&!R
95.20 S &=&-&↑3/3+&↑5/5-&↑7/7

96.01 S &1=1/10↑20!C TAN
96.10 I (&↑2=.01)96.2!S &=&/2;D 96!S &=2*&/(1-&↑2+&1)!R
96.20 S &=&+&↑3/3+&↑5/7.5+&↑7/315

97.01 C LOG
97.10 I (&↑2-2.04*&+1)97.2!S &=FSBR(99,&)!D 97!S &=2*&!R
97.20 S &=(&-1)/(&+1)!S &=2*(&+&↑3/3+&↑5/5+&↑7/7)

98.01 C EXP
98.10 I (&↑2=.01)98.2!S &=&/2;D 98!S &=&↑2;R
98.20 S &=1+&+&↑2/2+&↑3/6+&↑4/24+&↑5/120+&↑6/720

99.01 C SQUARE ROOT
99.10 S &1=&!S &=2!S &3=.000001
99.20 S &2=&1/&!I (FABS(&2-&)-&*&3)99.3!S &=(&+&2)/2!G 99.2
99.30 R

 C FSBR(90,ARG) OUTPUTS ARG IN 'E' FORMAT

90.10 S &1=0
90.11 I (&)90.12,90.9,90.2
90.12 T "-"!S &=-&
90.20 I (1-&)90,5!I (&-.09999999)90.7!G 90.9
90.50 S &1=&1+1!S &=&/10!G 90.2
90.70 S &1=&1-1!S &=&*10!G 90.2
90.90 T %1.05,&,"E",%1.&1!R
```

APPENDIX C

I WILL NOW GIVE SOME USEFUL INFORMATION FOR THOSE PEOPLE WHO HAVE AN ASSEMBLY LISTING OF FOCAL 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 RUBOUT PROCESSING ON THE CONSOLE DEVICE. STORE A ZERO THERE IF YOU HAVE A DEVICE WHICH IS NOT A CRT (SUCH AS A TELETYPE, DECWRITER, ETC.). IF YOU HAVE A CRT WHICH WILL BACKSPACE THE CURSOR WHEN SENT A 'BACKSPACE' CHARACTER (ASCII CODE 10 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 'EVMASK'. 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 IRQ 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 'JSR' 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 'C' BIT IS SET UPON RETURN FROM THE ROUTINE, FOCAL ASSUMES THAT AN INPUT ERROR OCCURED, AND ISSUES AN ERROR MESSAGE. (SEE CODE AT 'READC' 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 'JSR' INSTRUCTION AND WILL RETURN (VIA 'RTS') WITH THE 'C' BIT CLEAR IF NO ERRORS WERE ENCOUNTERED, AND THE 'C' BIT SET IF AN ERROR OCCURRED. THE ADDRESSES OF THESE ENTRY POINTS ARE PLACED IN THE DEVICE DISPATCH TABLES (SEE LABEL 'IDSPH'). THE RELATIVE POSITION IN THE TABLE DETERMINES THE DEVICE NUMBER OF THAT DEVICE. IF MORE THAN FIVE DEVICES ARE INSERTED, THE VALUES OF 'IDEVM' AND 'ODEVM' MUST BE UPDATED. THE ONLY PLACE THEY ARE REFERENCED IS AT 'CHKODV' AND 'CHKIDV'.

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 Ø. 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 'COMTAB' TABLE (THERE IS SPACE FOR HACKERS), AND THE ADDRESS OF THE ROUTINE TO PROCESS THE COMMAND IN THE 'COMADH' AND 'COMADL' 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 'FUNTAB' TABLE. THE ADDRESS OF THE ROUTINE TO HANDLE THE FUNCTION IS INSERTED INTO THE 'FUNADH' AND 'FUNADL' TABLES. LOOK AT THE CODE AT 'FUNC', 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 NORMALISED FLOATING POINT NUMBER IN 'FAC1' PRIOR TO RETURNING (BY JUMPING TO 'FPOPJ'.

HEED 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 'FRAN'). 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/OR ALTMODE 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' + A FEW TO WHATEVER ASCII CODE YOU WANT IT TO BE.

IF YOU 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 CONCUR' IN THE 'FCUR' FUNCTION (SEE LABEL 'FCUR').

<<<< NOTES >>>>