The NUMBER REVERSE Puzzle
DEVELOPMENT NOTES

This puzzle is an adaptation of "The Game of Reverse" by M. Burton as presented in *Forth Dimensions*, Volume 3 Number 5, January/February 1982, pp. 152-153.

Among other revisions, the program has been translated from fig-Forth into Forth-83, input error handling has been expanded, and the presentation format has been adjusted.

Block 1 is the Load Block. It supervises the loading of all the other blocks.

Block 2 includes the word **.r** which is copied from CF83-7, the Controlled Reference Words Set, Block 16.

It also includes the words **<builds** and **cls**, both of which are copied from CF83-8, the Uncontrolled Reference Words Set Plus, Blocks 82 and 101 respectively.

Block 3 provides the random number generator from CF83-11, the Sound, Joysticks, Timer, &, Random Numbers Words Set, Block 15.

Block 4 includes the variable **moves** which keeps track of how many reversals the player has played, the word **dim** which establishes the array of digits, and the word **y/n** which elicits a yes or no (y or n) response.

Block 5 provides the **instruct** word which presents the puzzle's instructions if the player indicates they are needed.

Block 6 dimensions the array of digits and provides the following words:

arr@ retrieves a digit from the array: n1 is the position index within the array and n2 is the digit returned.

arr! stores a digit n2 to the array at index n1.

arrayInit initializes the array.

arr. reports the current state of the array, i.e. the list of the nine digits in their current order.

Block 7 provides the words **arrayScramble** and **getInput**. The word **arrayScramble** randomizes the order of the nine digits in the array. Its step-by-step stack effects are:

```
: arrayScramble     --
  1                 1
  9                 1 9
  do                --
    i               i
    rnda            n                   0 <= n < i
    1+              n1                  n1 = i + 1
    i               n1 i
    arr@            n1 n2               n2 = digit at index i
    over            n1 n2 n1
    arr@            n1 n2 n3            n3 = digit at index n1
    i               n1 n2 n3 i
    arr!            n1 n2               n3 is stored at index i
    swap            n2 n1
    arr!            --                  n2 is stored at index n1
    -1              -1
    +loop ;         --
```

The word **getInput** elicits the entry of a digit between 0 and 9 inclusive.

An entry of 0 ends the try and returns the player to BASIC.

An entry of 1 works, but doesn’t actually reverse anything; it’s really just a NOP.

An entry of n (where `2 <= n <= 9`) reverses the n digits from the left of the list to digit number n. For example, if the list is currently:

```
2  5  7  6  9  8  1  4  3
```

And the player enters 6, then the new list will be:

```
8  9  6  7  5  2  1  4  3
```

The **getInput** word’s step-by-step stack effects are:

```
: getInput          --
  begin             --
    0               0
    cr .” Reverse how many? “
                    0
    pad             0 addr              addr = pfa of pad
    80              0 addr 80
    expect          0
          ( check if more than one character entered )
    span            0 addr2
    @               0 n
    1               0 n 1
    >               0 flag
    if              0
      ." Only 0 through 9 please.
                    0
    else            0
      pad           0 addr
      c@            0 char                character entered
      dup           0 char char
      dup           0 char char char
      48            0 char char char 48
          ( 48 = ASCII "0" )
      <             0 char char flag1
      swap          0 char flag1 char
      57            0 char flag1 char 57
          ( 57 = ASCII "9" )
      >             0 char flag1 flag2
      or            0 char flag
          ( flag = logical or of flag1 and flag2 )
      if            0 char
        cr ." Only 0 through 9 are allowed. "
                    0 char
        drop        0
      else          0 char
        48          0 char 48
        -           0 n1                  ( 0 <= n1 <= 9 )
        swap        n1 0
        1+          n1 1
      then          [ 0 or n1 1 ]
    then            [ 0 or n1 1 ]
  until ;           n1
```

Block 8 provides the words **arrayReverse** and **arrayCheck**. The word **arrayReverse** reverses the n digits to the left of the list. Its step-by-step stack effects are:

```
: arrayReverse      n
  dup               n n
  1                 n n 1
  >                 n flag                flag = true iff n > 1
        ( do nothing if n = 1 )
  if                n
    dup             n n
    2/              n n1                  n1 = n divided by 2
    1+              n n2                  n2 = n1 + 1
    1               n n2 1
    do                n
      dup             n n
      i               n n i
      1+              n n n3              n3 = I + 1
      dup             n n n3 n3
      arr@            n n n3 n4           n4 = array(n3)
      swap            n n n4 n3
      i               n n n4 n3 i
      arr@            n n n4 n3 n5        n5 = array(i)
      swap            n n n4 n5 n3
      arr!            n n n4              array(n3) = n5
      i               n n n4 i
      arr!            n n                 array(i) = n4
    loop              n n
    drop              n
  then                n
  drop ;              --
```

The **arrayCheck** word returns true if the array is in proper ascending numerical order. It returns false otherwise. It's step-by-step stack effects are:

```
: arrayCheck          --
  1                   1
  10                  1 10
  1                   1 10 1
  do                  1
    i                 1 i
    dup               1 i i
    arr@              1 i n               n = array(i)
    =                 1 flag1
          ( flag1 = true if the array entry = the array index )
          (   i.e. if the value is in its proper location. )
    and               flag
          ( 1 and 1 = 1; 1 and 0 = 0; 0 and 0 = 0 )
  loop ;              flag
```

Block 9 is the **reverse** word, the top controlling word of the puzzle. Its step-by-step functionality is:

```
: reverse
  randomize                Initialize the random number generator.
  cls                      Clear the screen.
  instruct                 Display title and ask if instructions
                                are needed. Display if yes.
  arrayInit                Initialize the array
  begin
    arrayScramble          Randomize the array
    0 moves !            Initialize the number of reversals.
    begin
      arr.               Report the list arrangement.
      getInput           Request the player’s command.
      dup 0=             Check for end of try.
      if
        1                End-of-Try flag
      else
        arrayReverse     Reverse the subset selected by player.
        1 moves +!       Increment the number of reversals.
        arrayCheck       Check if done.
      then
    until                Return if not done.
    arr.                 Report the list arrangement.
    ( msg )              Report number of reversals made.
    ( msg )              Ask if player wants to play another.
    y/n                  Get yes or no response.
    0=                   flag = true if answer is no.
  until
  bye ;                  Exit to BASIC
```

The Development Procedure involved:

1. Placing the CF83-W.dsk Working Disk in Drive 0 and entering **`RUN”CF83W`** .

2. Placing the NumberReverse.dsk in Drive 0 , entering **`1 Edit`** , and typing-in Blocks 1 through 9 (with all necessary testing, debugging, correcting, and re-formatting) .

3. Entering **`1 Load`** .

4. Placing the NumberReverseBIN.dsk in Drive 0 and entering:

```
        0 ‘ reverse savem NUMRVRSE.BIN
```

To try the puzzle:

From CF83, with the NumberReverse.dsk in Drive 0, enter **`reverse`** , or

From BASIC, with the NumberReverseBIN.dsk in Drive 0, enter **`RUN”NUMRVRSE`** .

** END **