Marvels of the FOR Command

by Winfried Gerum


Winfried Gerum

While some persons DO call languages like ALGOL, PASCAL or MODULA structured languages, they REPEAT that UNTIL people regard other programming languages as unstructured ones. Structured programming is possible under any programming language, even ASSEMBLER. The other way around for a careless programmer even in MODULA—which lacks the infamous GOTO—is to produce something like spaghetti code. It is a matter of education and personal discipline to produce structured code.

So FOR all MUMPS programmers I want to elucidate the aspects of the FOR command which is the most important control structure in this fascinating programming language. As so often in MUMPS there is simplicity: We have only one control structure for repeating code. And there is complex functionality: This single command can perform more tasks than the combined control structures in any other programming language. Let us compare FOR with the repetitive statements in PASCAL and C.

There are three such constructs. (P: stands for PASCAL, C: for the C programming language, M: for MUMPS; Code, expr ... stands for any code valid in the context)

while statement

```
P: while condition do Code
C: while (condition) Code
M: FOR  Code QUIT:'condition  Code
```

repeat statement

```
P: repeat Code until condition
C: do Code while !condition
M: FOR  Code QUIT:condition
```

for statement

```
P: for variable:= expr1 to expr2 Code
```

expr1 and expr2 must be of type integer. There is no way to give a stepwidth other than +1 or -1 (to = +1, downto = -1). The control variable cannot be altered within the loop.

```
C: for (var=expr1; var < =expr2;var++) Code
```

is equivalent to the above PASCAL code. However, C is more flexible: The triple of statements in the for-argument has no restrictions at all. From the view of the language syntax, there is no loop variable. Any of the three statements in the for-argument list may be empty. The first argument is executed once before entering the loop. The second argument is executed each time entering the loop. As long as the expression evaluates TRUE, the loop is repeated. The third statement is executed after each execution of a loop.

```
M: FOR var=expr1:1:expr2 Code
```

While the syntax of FOR is more powerful in C than in PASCAL

```
C: for (expr1; expr2; expr3 ) Code
```

there is always a MUMPS equivalent to the C-for:

```
M: XECUTE expr1 FOR  QUIT:'expr2  XECUTE
   Code:expr3
```

MUMPS is More

```
M: FOR  Code Q:'condition  Code2
```

has no elegant equivalent in PASCAL

```
P: Code1 while condition do Code2 Code1
```

which repeats Code1 in the routine text. It is possible to avoid the repetition of Code1 using a GOTO. But PASCAL programmers are not supposed to use GOTO!

```
C: for (;;) {Code1; if (!condition) break;
   Code2 }
```

that is essentially the same as in MUMPS, but the empty for argument looks somewhat strange.

While the following MUMPS constructs have some kind of equivalent in PASCAL and C,

```
M: FOR var=numexpr1:numexpr2:numexpr3 Code
```

with numexpr2 not equal +1 or -1

```
M: FOR var=numexpr1:numexpr2 Code
```

the next constructs are unique to MUMPS:

```
M: FOR var=numexpr1:numexpr2:numexpr3,nume
   xpr4:numexpr5:numexpr6 Code

M: FOR var=expr1,expr2,expr3 Code
```

Especially if you consider that the last variant does not restrict the values of var even to numerics, any string value is possible.

Try to write an equivalent to

```
FOR Person="Frans","Joop","Marcus" WRITE !
,"Good morning ",Person
```

in any other language and you know why we *love* MUMPS!

Back to MUMPS

Let us have a systematic look at all the possibilities!

All variants have in common that the range of the command is the rest of the line that contains the FOR command. In former times, that has been a restriction limiting the value of the FOR command. But the introduction of the block syntax with the argumentless DO essentially did away with that restriction. All variants may have a terminating command: a QUIT or GOTO or HALT. Note that with multiple FOR commands in one line, a QUIT command QUITs just one level, while GOTO (and HALT of course) terminates all FOR commands nested in one line.

The argumentless form

```
FOR  Code
```

and the open ended form

```
FOR var=numexpr1:numexpr2 Code
```

repeat forever unless a terminating command gives a way out.

Note that the form

```
FOR var=numexpr1:numexpr2:numexpr3 Code
```

and

```
FOR var=numexpr1:numexpr2 Code
```

take the numeric interpretation of begin, stepwidth and limit-value, while

```
FOR var=expr
```

does not make any special interpretation of expr. The values of their expressions are evaluated once before the loop is entered and before the loop variable is assigned a value.

So code like

```
+1 ;compute factorial N
+2 SET X=1 FOR N=1:1:N SET X=X*N
+3 Q
```

may look strange to some people. But it is absolutely reasonable code: Provided N is a positive integer, this variable leaves the loop with the same value it had at its entry!

On the other hand, code like

```
+1 FOR I=1:1:BUDGET DO SPEND IF ELECTION(1
    ) SET:ELECTIONS(I) BUDGET=BUDGET*2
+2 Q
```

probably does not perform as intended: FOR does not exceed the original Budget!

The FOR variable may be freely changed by the code invoked by the FOR command (do it only if reasonable and well-documented or obvious). FOR must even detect a KILL on the FOR variable and trigger an error. So the introduction of an error trapping mechanism in MUMPS—which is long overdue—would give a quick and dirty way to terminate a FOR command even from within a subroutine or a block-DO by killing the FOR variable.

Saving Some Characters

Programmers who abhor unnecessary codes will be happy to learn that code like

```
+1 SET FROM=2*N,TO=X*N
+2 FOR I=FROM:1:TO Code
+3 Q
```

may be written even more concisely than

```
+1 FOR I=(2*N):1:(X*N) Code
+2 Q
```

since the brackets are simply superfluous in MUMPS. Try

```
+1 FOR I=2*N:1:X*N Code
+2 Q
```

It is completely intelligible for your interpreter (and for all colleagues reading "Tips 'n' Tricks").

A Big Example

The various nonargumentless forms of FOR may be combined into complex statements such as

```
+1 SET LIMIT=50
+2 FOR P="List of all primes below",LIMIT,
   2,3:P:18,P:2 W !,P Q:P>LIMIT  IF P,$$PR
   IME(P) W " Prime"
+3 Q
```

This is almost an all-in-one example of the power of FOR. The first FOR argument gives the caption of a list of numbers. The second argument is the only even prime (some say "2 is the oddest prime"). Subsequent tests for primality need not check even numbers.

The third argument evaluates when it is entered to 3:2:18. The fourth argument evaluates to 17:2 when due. This is because the upper limit of the previous argument is not reached exactly, and an increase beyond the limit is not done. The number 17 appears twice in the list. Complex FOR statements like these are not treated as nested FOR statements concerning a terminating QUIT statement.

Caveat Programmator

We see that a FOR argument list is not quite the same as the argument list of other commands. Therefore it is no surprise that

```
FOR @X Code
```

is not allowed, while

```
FOR @X=@Y Code
```

with @X evaluating to a local_variable_name and with @Y evaluating to a for_argument_list

and

```
FOR I=@X:@Y:@Z Code
```

with X,Y,Z evaluating to variable_names should be processed properly by all MUMPS implementations. A novice programmer might want to cleverly condense

```
FOR I=1:1:10 FOR J=1:1:10 SET A(I,J)=""
```

to

```
FOR I=1:1:10,J=1:1:10 SET A(I,J)=""
```

which is valid syntax, but very different in semantics: J is not a loop variable in the latter example. J=1 is treated as a (boolean) expression, yielding either 0, 1 or <UNDEF>.

Counting With FOR

As most benchmarks—whether clever or not—rely heavily on the FOR command, it is reasonable to expect some kind of special tuning. Therefore, code like the following, counting the subscripts of an array

```
+1 SET CNT=0,X=""
+2 FOR  SET X=$O(ARRAY(X)) QUIT:X=""  SET
   CNT=CNT+1
```

can be expected to run considerably slower than

```
+1 SET X=""
+2 FOR  SET CNT=0:1 SET X=$ORDER(ARRAY(X))
    QUIT:X=""
```

The Story Goes On

If you think that is all about FOR, look at the MUMPS standard of the year 2001. There you might find

```
FOR intexpr Code
```

which repeats Code intexpr times without concern about a loop variable. ❖

This column is reprinted from MUG-Europe Newsletter, *Vol. VIII, No. 2/3, 1991.*

Winfried Gerum is president of Winner Software which he founded in 1991. The company—located in Erlangen, Germany—is dedicated to providing the MUMPS community with tools, consulting, and related services. Gerum has been active in the MUMPS community for many years, writing articles, contributing proposals to the MDCC-Europe, and helping to organize the 1991 MUG-Europe conference in Numemberg—his birthplace.
