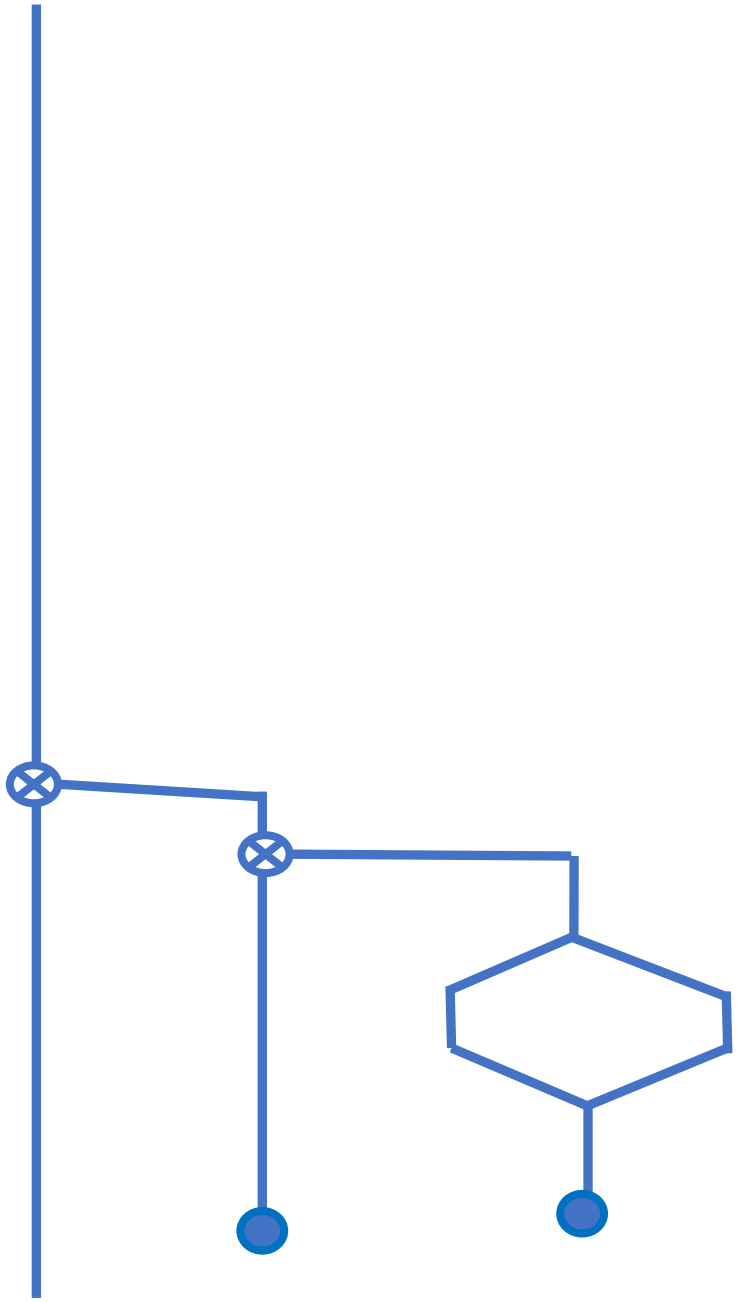


Living In The Matrix

SVFIG
Dec. 20, 2025
Bill Ragsdale

Credits

- Andrew McKewan and Tom Zimmer for Win32Forth.
- The European team who updated it in the early 2000s.
- Dr. Julian Noble for the key concepts of matrix structure and its parameter use. See *Scientific FORTH (But for early DOS.)*

Last Month

- We examined an OOP object design for matrices, bottom up.
- We considered its methods as if silicon assembly language op-codes.
- This month we'll view top down, for the syntax the programmer sees.
- And a number of high level matrix operators.

Today We'll Cover

- Traditional matrix syntax.
- A Forth appropriate syntax.
- A Forth matrix extension word set.
- Operation on matrices.
- Matrix calculations.

Traditional Matrix Syntax

```
C[1;3] = A[2;2] + B[2;3]
```

A variable name

Cells within the matrix

Data operations on the values

The Good and the Bad

Traditional syntax is terse. But . . . .

Not clear in cases such as:

```
C[ :  ; 3:end] = A[ : ; 1 2 3](3 :
2) + B[1 2 ; 3 4]
```

Forth factors out the nesting and has a clear reverse Polish ordering.

Matrix Forth Levels

Level Zero — OOP Matrix Object
RCaddr: (r c --- addr)

Level One — Cell to Cell (r c --- addr)
}} }}@

Level Two — By rows/columns/sub areas
}xCyCcopy (x{ c1 y{ c2 ---)

Level Three — Matrix level
matrix{ }fill }list }copy
} + }= }transpose}dot*

Our Syntax, Summary

Matrix data are ANSI 64-bit floats.

Matrix names end with '{' Demo{

Matrix operations begins with '}' }Fill

Cell operators begin '}}'

Pi 4 5 }}! 4 5 }}@ see 3.14

Basically, that's it!

Review

Forth Matrix is zero based for rows & columns.

Words with { generate a matrix address.

Words with a double }} with row and column specifiers operate on a cell.

x{ 2 3 }}@

Words with } operate on a matrix address:
first{ }list

Picking It Apart

```
A{  2 3 }} F@     \ or }}@
```

In Forth:

A{ gets the address of a matrix.

2 3 specify the row and column.

}} gets the storage address.

F@ an operation on that address.

Or, combined into }}@.

Comparison

Traditional Matrix Syntax

```
C[1;3] = A[2;2] + B[2;3]
```

In Forth:

```
A{ 2 2 }}@
B{ 2 3 }}@  F+
C{ 1 3 }}!
```

Creating Float Matrix

```
matrix{ 3 4  first{
 first{   }list
```

```
.00000 .00000 .00000 .00000
.00000 .00000 .00000 .00000
.00000 .00000 .00000 .00000  ok
```

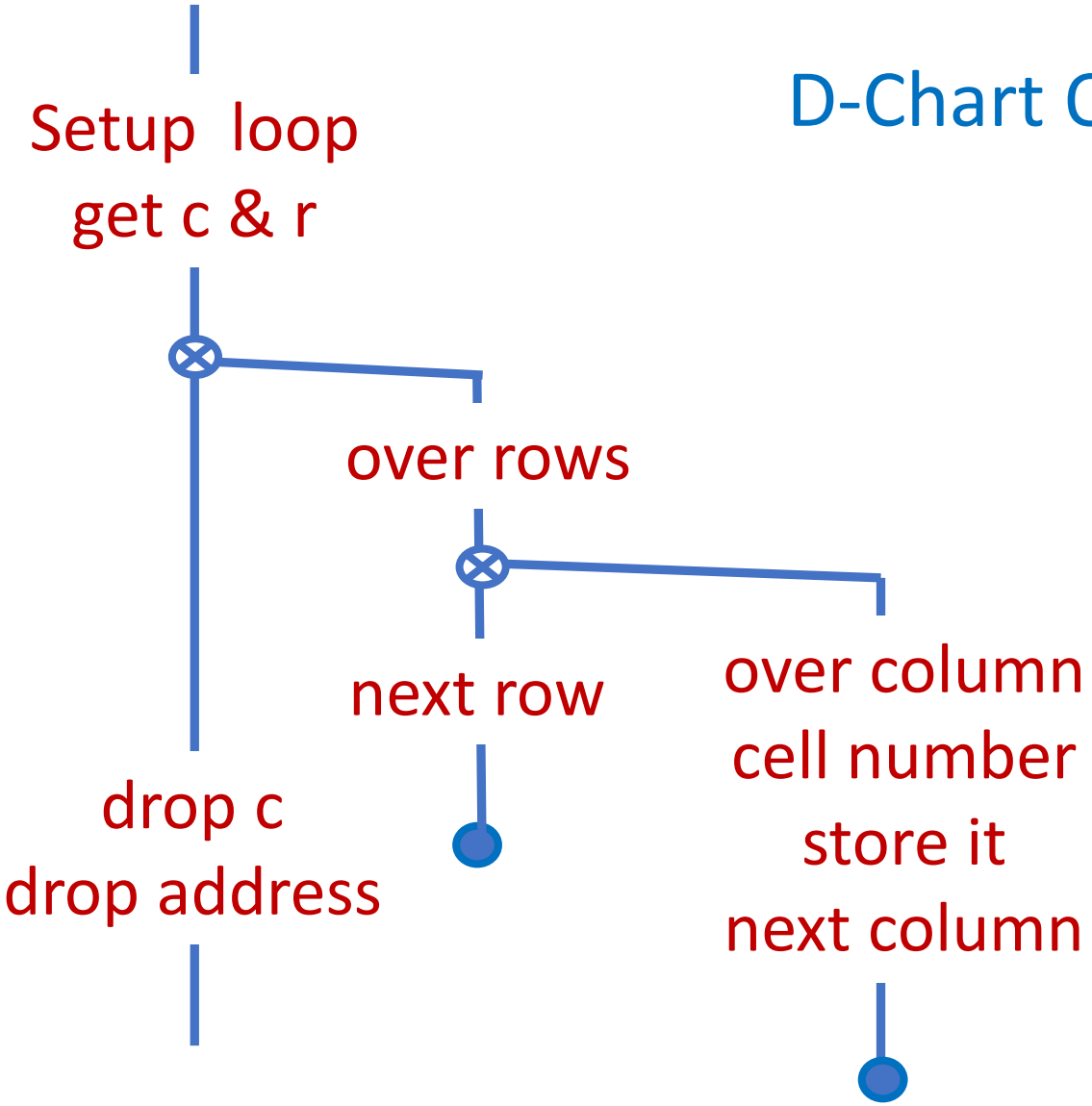
Quick Entry For Testing

```
first{  }fill

first{  }list


.00000 1.0000 2.0000 3.0000
4.0000 5.0000 6.0000 7.0000
8.0000 9.0000 10.000 11.000  ok
```

Programming Example

```
: }Fill  ( matrix_address  ---  )
   dup }RC swap ( c r )
 0 DO
   dup 0  DO
          dup j * i + s>f <<stored value
         over j  i }}!    <<storing
        LOOP
   LOOP
   2drop   ;
```

D-Chart Of }fill



A Code Example, L0 & L3

```
:M List: ( x{ --- )
 Name: self count type CR
 #rows 0 do
      #cols 0 do
        j i RCget: self  f.
    loop   cr
  loop  ;M

: }List  ( x{ --- )  List:  :Matrix{  ;
```

Direct Data Entry

```
first{ {[ 1 1 1 1 | 2 2 2 2 | 3 3 3 3 ]}

first{  }list

1.0000 1.0000 1.0000 1.0000
2.0000 2.0000 2.0000 2.0000
3.0000 3.0000 3.0000 3.0000  ok
```

Matrix Math

```
first{ first{ first{  }+
              first{  }list
that is: A  B  +  C !
```

```
 .0000 2.0000 4.0000 6.0000
8.0000 10.000 12.000 14.000
16.000 18.000 20.000 22.000
```

Note: An intermediate matrix allows overlapping operations.

Transposing A Matrix

```
first{ }fill    first{ }list
 .0000  1.0000  2.0000  3.0000
4.0000  5.0000  6.0000  7.0000
8.0000  9.0000 10.0000 11.0000


first{ }transpose    first{ }list
.00000  4.0000  8.0000
1.0000  5.0000  9.0000
2.0000  6.0000 10.0000
3.0000  7.0000 11.0000
```

Transpose Example

```
: }Transpose   ( x{ --- )
  Temp{ over }RC }Initialize   \ a temp
  dup Temp{ }Copy  \ copy to temp
  dup dup }RC swap }Initialize
  Temp{ }RC             \ get rows and cols
    0 do  dup 0 do
      Temp{ i j }}@  over  j i }}!
      loop loop 2drop
   Temp{ }Release  ;
```

Matrix Product Example

```
: }.*  ( x{ y{ z{ --- )
   {:  x{  y{  z{ |  xC  zR  zC :}
 x{ }RC  to xC  to zR
 y{ }C   to zC
 z{ zR zC  }Initialize \ product matrix
 zR  0 do  zC  0  do  xC   0 do
       x{ k i }}@       \ from left matrix
       y{ i j }}@  F*   \ from upper matrix
       z{ k j }}+!      \ into z{
   loop  loop   loop  ;
```

Matrix Product Example

```
                                  B3X2{
                                    1.00      2.00
                                    3.00      4.00
                                    5.00      6.00

A2X3{                                       C2X2{
    1.00      2.00      3.00          22.00     28.00
    4.00      5.00      6.00          49.00     64.00
```

Word Summary

Create: matrix{

Support: }list }fill

Data Entry: {[|]}

Cell Math: }} }}@ }}? }}! }}+!

Row/Col: }}rrExch }}ccExch

Matrix Math: }+ }- }* }/ }dot*

Manipulation: }transpose }invert }det }eye

Summary

I use MatLab and the clone Octave.

I have a project about once a year and need a complete refresher on the semantics.

The manuals explain each single structure but not the overall parsing order.

Therefor I need a reverse Polish syntax that needs no ‘refresher course’.

Matrix Forth is the answer.

References

- https://github.com/BillRagsdale/Matrix-Forth-Wordset
- https://github.com/BillRagsdale/WIN32Forth-Guide