3d.matrix – A FORTH Word To Create 3D Matrices

The goal was to build a word which creates 3D matrices. The usage is:

a b c 3d.matrix m

where a, b and c are the maximal numbers of the 3 dimensions of the new matrix m. I.e. a value of 2 for a means that the first address parameter to determine a certain cell of the matrix allows for using 0 and 1 as indices. Hence, an address of 1 2 3 for the above declared matrix is fine, whereas 2 2 3 is not: a is a higher index than the matrix is designed for.

Design Goals

As depicted above already there should be bounds-checker to prohibit the user from entering parameters which will lead to segmentation faults or overwriting the dictionary or other valuable things in memory.

When m is called and the stack is empty it shall depict the following:

```
m
     PFA = 140070340884984
   c (z) = 4
   b (y) = 3
   a (x) = 2
1st cell = 140070340885008
 ok
```

The 1st line shows the parameter field address which can be considered as the mother address of all data concering the matrix m. c, b and a are the numbers of dimensions which means, c is the dimension of z, so legal indices of z are 0, 1, 2 and 3. And so forth. The last line is the address of the 1st user cell, this is the first cell which can contain a payload. So, called with an empty stack m tells you all the technical data you need to use it correctly.

If three parameters are on the stack when m is called, and these parameters are within the legal constraints, a write process can be launched using ! , or a read process can be launched using @ or ? .The command

1 2 3 m

will leave nothing on the stack but the cell address.

If the stack contains 1 or 2 values when m is called, it will output an error.

Another Goal

3d.matrix was built without using any outside words, in other words, it uses only standard FORTH words. My personal test was as follows:

I put 3d.matrix in a single file containing nothing but the sourcecode of 3d.matrix. Then I tested it in Gforth:

gforth 3d.matrix.f

… and in SwiftForth:

sf 3d.matrix.f

Both FORTH variants work with this code without complaints or warnings. This was a goal I wanted to achieve.

Mathematics

In order to address a linear sequence of memory cells in a 3D address format you have to use some math. In case of a 2 3 4 matrix we have 24 cells with indices running from 0 through 23.

1. step: Divide into two segments, each 12 cells long: 0…11 and 12…23

2. step: Divide into six segments, each 4 cells long: 0…3, 4…7, 8…11, 12…15, 16…19, 20…23

3. step: All done! The third range is one cell at every index of the 24 indices available.

To make things short and concise:

The formula for the offset is

$cyx + bx + a$

Herein you recognize a, b and c as the cell address values and x, y and z as the boundaries of our matrix. Offset? Yes, you have to add the result of this formula to the baseaddress, which is PFA plus 3 cells for x, y and z plus the offset we just calculated.

The Source

The code contains hints which refer to the last chapter of
this document. Please, don't overlook them!

```
: 3d.matrix
  create
    ( a b c )
    dup
    ,
    swap
    dup
    ,
    rot
    dup
    ,
    *
    *
    3 + \ We must allot 3 cells in addition to store a, b and c!
    allot \ See last chapter, no II
  does>
    ( a b c pfa )
    \ Stack check
    depth 1 = \ no parameters, just <matrix> = pfa \ See last chapter, no I
    if
      cr                    \ .pfa.        Values/Variables in .x. notation are addresses!
      dup                   \ .pfa. .pfa.
      ."      PFA = " . cr  \ .pfa.
      dup                   \ .pfa. .pfa.
      @                     \ .pfa. pfa
      ."    c (z) = " . cr  \ .pfa.
      dup
      1 cells +
      @
      ."    b (y) = " . cr
      dup
      2 cells +
      @
      ."    a (x) = " . cr
      3 cells +
      ." 1st cell = " . cr
      exit
    then
    depth 4 <
    if
      ." not enough data"
```

```
   exit
then

\ START OF REGULAR OPERATION:

swap         \ a b .pfa. c
2swap        \ .pfa. c a b
rot          \ .pfa. a b c

\ Check limits:

dup          \ .pfa. a b c c

4 pick @     \ .pfa. a b c c c.limit
1 -
\ cr ." c : " prstln
> if ." c off limits" abort then

swap         \ .pfa. a c b
dup          \ .pfa. a c b b
4 pick
1 cells + @ \ .pfa. a c b b b.limit
1 -
\ cr ." b : " prstln
> if ." b off limits" abort then

rot          \ .pfa. c b a
dup          \ .pfa. c b a a
4 pick
2 cells + @ \ .pfa. c b a a a.limit
1 -
\ cr ." a : " prstln
> if ." a off limits" abort then

\ End of check sequence!

             \ .pfa. c b a
swap         \ .pfa. c a b
rot          \ .pfa. a b c
dup          \ .pfa. a b c c
4 pick       \ Ermittle y
1 cells +    \ ... für y
@            \ .pfa. a b c c y
*            \ .pfa. a b c cy
4 pick       \ Ermittle x
2 cells +    \ ... für x
```

```
    @           \ .pfa. a b c cy x
    *           \ .pfa. a b c cyx
    rot         \ .pfa. a c cyx b
    4 pick      \ Ermittle x
    2 cells +   \ ... für x
    @           \ .pfa. a c cyx b x
    *           \ .pfa. a c cyx bx
    rot         \ .pfa. a cyx bx c
    drop        \ .pfa. a cyx bx
    +           \ .pfa. a cyx+bx
    +           \ .pfa. a+cyx+bx
    +           \ sum
    3 cells +   \ This is to find the first payload cell address
;
```

Development Environment

The development took place on a small ASUS Zenbook 14 note-book running Kali Linux. Main work was done with Swift-Forth, last tests included Gforth, too. Please, use Gforth 0.7.9_20231116! You'll find the download link on their homepage, and don't get fooled by the statement "current version"…

Questions And Remarks

I

```
    depth 1 = \ no parameters, just <matrix> = pfa \ See last chapter, no I
```

This line works with the value 1 as shown but also with the value 0. I.e. in case your stack is empty and you call m, it will work either way. I don't understand why this is so. It seems as if m placed some ghost value on the stack.

II

The create part of the word, which is executed during compile-time, uses ALLOT as the last command. This is the way many FORTH commanders seem to prefer.

My question is: Why? Why don't you ALLOT first and use the , (comma) afterwards?

Maybe because the compiler takes care of the values and doesn't write them into illegal space?

My Motivation

The rationale or motivation for the project was to rigorously understand all the details one has to know about CREATE and DOES>.

This was a success :)

Nevertheless, you'll never stop learning, there will always be questions and dought which drive you further. So, I appreciate every thoughtful comment, hint and even critique ;)