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 concerning 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 ? .

I did explicitly not want to mimick syntax symbols from

other languages like brackets. My goal was a FORTHish way to solve the problem.

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

(azy + bz + c) * cellsize

or

```
azy   \ a * z component * y component
bz    \ b * z component
c
+
+
cells \ multiply by cell size
```

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 +   \ Drei Zellen mehr allotieren!
    4     \ Cell size! allot works with bytes!
    *     \ So we have to multiply it!
    allot
  does>
    ( a b c pfa )
    \ Stack check
    depth 1 =  \ no parameters, just <matrix> = pfa
    if
```

```
  cr                    \ .pfa.
  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:
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 -
> 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 -
> 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 -
    > if ." a off limits" abort then

    \ End of check sequence!

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

    \ target formula: azy + bz + c

                \ .pfa. a b c
    rot         \ .pfa. b c a
    dup         \ .pfa. b c a a
    4 pick      \ Fetch z
    @           \ .pfa. b c a a z
    *           \ .pfa. b c a az
    4 pick      \ Fetch y
    1 cells + @ \ .pfa. b c a az y
    *           \ .pfa. b c a azy
    swap        \ .pfa. b c azy a
    drop        \ .pfa. b c azy
    rot         \ .pfa. c azy b
    3 pick      \ Fetch z
    @           \ .pfa. c azy b z
    *           \ .pfa. c azy bz
    +           \ .pfa. c azy+bz
    +           \ .pfa. c+azy+bz
    cells
    +           \ sum up
    3 cells +   \ Find first payload cell
;
```

Development Environment

The development took place on a small ASUS Zenbook 14 notebook running Kali Linux. Main work was done with SwiftForth, 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 as intended with the value 1. I.e. in case your stack is empty and you call m, it will show you the technical data.

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 :)

And to help other FORTH apprentices, I've made list of things which I stumbled upon:

1. Don't take addresses for values and vice versa! I've invented the .a. format to differentiate between a as an address which would be written as .a., and a as a value you'll receive by

   ```
   .a. @
   ```

2. Make a protocol of each stack state, i.e. make a comment after each word (command) showing what you think is the state of the stack after executing the word (command). Check your assumptions against brutal reality.
3. I spent nearly two days (about 16 work hours) to… No, can't say this, you'll consider me a moron.

4. Spend enough time on the mathematical background. Write your own recipes and formulae, work on this until you have understood each and every step.
5. Don’t give up!

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

Annotations

Last version: 2023-11-22@1700 (Germany, CET/MEZ)

Author: HMB alias Dude_McGeex (Reddit) alias DudeMcGee (Github.com)