# Extending the VALUE concept

## Extending the VALUE concept

*VALUEs are better than VARIABLEs*

* Makes the code a bit simpler (slightly)

** Can be initialised at compile time (only occasionally useful)

*** You don’t need @ or ! any more

**** Compatibility with locals

Better – but only a bit better

# Extending the VALUE concept

*Modifiers, not operators*

*123 → MYVALUE*

Forth operators come after, modifiers come before

New terminology makes Forthers feel happier

Better – but only a bit better

# Extending the VALUE concept

## *Enumerate modifiers*

```
ENUM VALMODS {
  VMOD@      \ Fetch
  VMOD!      \ Store
  VMODADDR   \ Address
  VMODINC    \ Increment
  VMODDEC    \ Decrement
  VMOD+!     \ Add
  VMODOFF    \ Zero
  VMOD-!     \ Subtract
  VMODSIZE   \ Size of
  VMODSET    \ Set
  VMNUMMODS  \ Number of modifiers
};
```

0 = The default - unmodified

Quite a lot better!

**SSMNIC**

The Society for the Suppression
of
Magic Numbers
in Code

# Extending the VALUE concept

## *Different data types*

Float is the only useful one

```
1.2E3 FVALUE MYFVAL  ok
MYFVAL F. 1200.  ok
4.56E7 -> MYFVAL  ok
```

Not much is saved by doing smaller sizes of single elements

# Extending the VALUE concept

## *Indexed values*

Much more useful

```
: VINDEX nsize "name" -- ;
Exec: (unmodified) nindex – ncontents

10 VINDEX MYINDEX  ok
12 1 -> MYINDEX  ok
34 2 -> MYINDEX  ok
1 MYINDEX . 12  ok
2 MYINDEX . 34  ok
1 SIZEOF MYINDEX . 4 ok
```

Size of element, not size of array

Automated address calculation
All modifiers available

# Extending the VALUE concept

## *Indices: Zero-based or one-based?*

Zero for programming nerds, one for normal folk

```
10 0 -> MYINDEX  ok
100 10 -> MYINDEX  ok
0 MYINDEX . 10  ok
10 MYINDEX . 100  ok
```

VINDEX allows both

# Extending the VALUE concept

## *Indices: Programming rule*

Modifier always goes immediately before the VALUE to be modified

`2 1 -> MYINDEX` and `2 -> 1 MYINDEX` have the same effect.

However, in practice, the index itself is often a VALUE, and the statements

`12 MYVAL -> MYINDEX` and `12 -> MYVAL MYINDEX` do not have the same effect.

# Extending the VALUE concept

## *Indices: Trapping index bounds*

Miscalculated index a prime source of programming errors

```
1 11 -> MYINDEX
Invalid index 11 for MYINDEX length 10
 ok
```

VINDEX constrains the index
Reports the error on the terminal, if in debug mode
Reports on the Zlog file, with source file, word and line number, if in run mode.

# Extending the VALUE concept

## *Two dimensions: VMATRIX*

```
: VMATRIX nsizex nsizey "name" -- ;
Exec: (unmodified) nindexx nindexy -- ncontents
```

This operates in exactly the same way as VINDEX but with two indices.

```
10 20 VMATRIX MYMATRIX  ok
12 3 4 -> MYMATRIX  ok
3 4 MYMATRIX . 12  ok
```

It also checks for valid indices:

```
1 10 21 -> MYMATRIX
Invalid index 10 21 for MYMATRIX length 10 20
 ok
```

# Extending the VALUE concept

## *Indexed strings*

```
: STRINDEX narraysize nmaxlen "name" -- ;
Exec: (unmodified) nindex -- addr

10 100 STRINDEX MYSTRS  ok
Z" abc" 1 -> MYSTRS  ok
Z" xyz" 2 -> MYSTRS  ok
1 MYSTRS Z$. abc ok
2 MYSTRS Z$. xyz ok
1 SIZEOF MYSTRS . 100  ok
```

→ does the string store.

*To do: Constrain string length as well as index*

# Extending the VALUE concept

## *The problem with structures*

a) Elements can be any size (byte, word, int, longlong, string, another structure)

b) Therefore, you need to remember whether to use C@, W@, L@ etc.

c) Naturally, you sometimes forget

# Extending the VALUE concept

## *The solution for structures*

Use value elements instead!

```
: VFIELD structlen size "name" -- structlen' ; ??? -- ???
```

defines a value type field of arbitrary length.

The following words are added for all standard sizes:

```
: VBYTE     _BYTE       VFIELD  ; \ Value type byte field
: VWORD     _WORD       VFIELD  ; \ Value type word field
: VINT      _INT        VFIELD  ; \ Value type int field
: VLONGLONG _LONGLONG   VFIELD  ; \ Value type longlong field
```

N.B.
Linux32 LONG=32
Linux64 LONG=64
LONGLONG=64 in both
Therefore, always use
LONGLONG, never LONG

# Extending the VALUE concept

## *Values in structures - example*

```
STRUCT MYSTRUCT
  VINT         my1
  VWORD        myword
  VBYTE        mybyte
  100  VFIELD  mystring
END-STRUCT

MYSTRUCT BUFFER: MYSTR

: TESTER
  123 MYSTR -> my1
  45  MYSTR -> myword
  67  MYSTR -> mybyte
  Z" qwerty" MYSTR -> mystring
  MYSTR my1 .
  MYSTR myword .
  MYSTR mybyte .
  MYSTR MYSTRing z$. SPACE
  MYSTR SIZEOF my1 .
  MYSTR SIZEOF myword .
  MYSTR SIZEOF mybyte .
  MYSTR SIZEOF MYSTRing .
;

TESTER 123 45 67 qwerty 4 2 1 100  ok
```

# Extending the VALUE concept

## *Dynamically created values*

*: ZVALUE z$name  ival -- ; Exec: (unmodified) -- val*

Needed for values that are created automatically during compilation, by reading in a GLADE file.
See “In Cahoots - Forth, GTK and Glade working secretly together” (Euroforth 2017)
http://www.euroforth.org/ef17/papers/nelson.pdf

```
: ZVINDEX nsize z$name -- ; Exec: (unmodified) nindex -- ncontents
: ZSTRINDEX narraysize nmaxlen z$name – ; Exec: (unmodified) nindex -- naddr
```

Needed for indexed values that are created automatically during compilation, by reading in an SQL table.
See “A radical Forth alternative to the Windows registry”
To be presented later.

# Extending the VALUE concept

*Whizzo new idea!*
*Automatic creation of Locals from SQL result*

Imagine...

```
MAXOPERATORS VINDEX   OPNUMS
MAXOPERATORS STRINDEX OPNAMES

: GETOPS ( --- ) \ Get operator names & nos. from SQL into memory
  SQL| SELECT opnum,opname FROM OPERATORS |SQL> IF    \ Run query
    PROWS 0 DOROW                                    \ Each row
      ropnum  I -> OPNUMS                            \ Save no.
      ropname I -> OPNAMES                           \ Save name
    LOOP
  THEN
;
```

Wanted:
VFXpert to implement dynamic locals

Ropnum and ropname are locals created and assigned automatically from the SQL result.