Eine Sprache selbst erfinden – anstatt sie zu erklaeren **Teil 1 v16 ExMark March 2018**

Als Grundlage definieren wir einen Teil echter Hardware, werden diesen aber auf dem PC komplett in Software implementieren. Wir ersetzen Hardware mit PC, Tastatur und Bildschirm. In dieser Minimum-Version wird nur die PW-LED links simuliert. Alle 16 LEDs werden im spaeteren Beispiel **35 Forth-Worte** kontrolliert.

Die echte Hardware besteht aus der PW LED, 3 Tastern, 4 OUTPUT LEDs, 4 INPUT LEDs und 4 zusaetzlichen LEDs; jeder dieser “Pins” kann dann spaeter auf HIGH oder LOW, 1 oder 0 gesetzt werden – und dies entspricht dann den 16 LEDs, die den Status widerspiegeln. Die 4 O, I und A Werte stellen binaere Werte dar, und koennen damit als Einzelbits oder auch die 4 Bits eines Nibbles eingesetzt werden, z. B. als Werte eines Zaehlers.

```
PW  T3 T2 T1        O3 O2  O1 O0        I3 I2  I1 I0       A3 A2  A1 A0
 1   1  0  0         1  1   1  1         0  1   0  1        1  1   1  1
```

Als ersten Schritt wollen wir erreichen, dass wir PW auf 1 und 0 setzen koennen. Aehnlich bei anderen LEDs.

Aber die Beschreibung, wie dies programmiert wrrden muss, ist oft schwierig, denn es ist immer der falsche Level. Als Alternative stellen wir uns einfach ganz dumm, fangen an, denken nach und gehen durch den Prozess. Stellen wir uns einfach vor, dass wir gern “eine neue Computersprache erfinden” wollen – wenigstens die Grundlagen. Hier wollen wir nur sehr einfache Konstrukte einsetzen die jeder versteht – komplexer geht immer und spaeter. Alle normalen ASCII Zeichen koennen eingesetzt werden und sind fuer die Kommandodefinitionen (Worte) erlaubt.

Alle diese Worte muessen voneinader getrennt werden – und wir verwenden eine Leerstelle, genauso wie bei nomalem Schreiben. Das entsprechende englische Wort ist SPACE. Und wir nennen sie wie in Forth ueblich - Worte

SPACE - 1 und damit haben wir unser erstes Wort definiert.

Programmieren wird eine unserer Aktivitaeten sein – aber wir muessen sicherstellen, dass wir den Code auch spaeter verstehen oder erklaeren koennen – also fuegen wir Kommentare hinzu, als Erklaerung was der Code tut. Der Computer versteht das nicht, also muessen wir zwischen beide Teile eine “ Mauer” einbauen. Die Tastatur gibt uns zwei einfache Optionen: \ und /. Die Division verwendet / also ergibt sich als Antwort: \ . Dieses ASCII-Zeichen zeigt dem Computer, dass der Rest der Zeile bis zum Ende ein Kommentar ist und vom Computer zu ignorieren ist.

**SPACE \ - 2 Worte sind es dann jetzt – SPACE und **

Viele Programmieraspekte muessen abgedeckt werden; z.B. muessen wir ASCII-Text an den Bildschirm schicken. Um dies zu definieren – setzen wir den Text in “ Quotes” . Ein Kommando muss definieren, dass Text auf den Bildschirm geht. Der Punkt “.” erreicht das. Kombiniert mit den Quotes ist es die Loesung. Damit koennen wir schon das .” HELLO WORLD “schicken. Versuchen Sie es. Ein .“HELLO WORLD “ eingetippt – aber nichts passiert. Nun, der Computer weiss ja nicht, dass wir mit der Eingabe fertig sind; das sagt man ihm mit der Return /Enter / <CR>-Taste. Also jetzt <CR> gedrueckt nach dem .” HELLO WORLD “ <cr>, und Leerstellen nicht vergessen. Jetzt kommt die Antwort auch HELLO WORLD. Wir koennen, womit andere Sprachen anfangen. Der Punkt allein “.” kommt spaeter.

SPACE \ . .” xx “ - von 2 vorher sind wir jetzt bei 5 Worten angekommen.

Wie kann man denn bekannte Worte kombinieren, und denen einen neuen Namen geben? Wir muessen dem Computer sagen, wo es anfaengt, den Namen, was enthalten ist, wo es aufhoert. Beispiel ***: HALLO .” Hello World “ ;*** Der **“:”** startet die Definition, dann der Name **HALLO** die Kombination **.”Hello World “** und **“;”** beendet die Definition

SPACE \ . .” xx “ : ; - zu den 5 Worten sind 2 dazugekommen, also jetzt 7 Worte.

Dieses “HELLO WORLD” Beispiel weitergefuehrt: Wir definieren ein neu kombiniertes Wort : DISPLAYIT .” HELLO WORLD “ ; Auf <cr> antwortet der Computer mit OK, was heisst intern verarbeitet und alles OK. Eingabe DISPLAYIT <cr> , und HELLO WORLD erscheint auf dem Bildschirm. Noch einmal dasselbe und ein weiteres HELLO WORLD folgt dem ersten. Und wie kriegt man die untereinander? Ueber die Tastatur ist es <cr>, also CR definiert als neues Wort.

SPACE \ . .” xx “ : ; CR - und damit erhoeht sich der Wortschatz von 7 auf 8

Wir modifizieren unser : DISPLAYIT .” HELLO WORLD “ ; auf : DISPLAYITCR .” HELLO WORLD “ CR ; <cr> und wir haben was wir brauchen. Der PC antwortet mit ok ein neues Wort ist erschaffen. Ausprobiert mit DISPLAYITCR <cr> und es erscheint, das 2. Mal, unter dem ersten Mal und so weiter.

Zusaetzlich sollte es moeglich sein, den Bildschirm leerzuwischen und wieder oben links anzufangen, sozusagen mit einer sauberen Seite anzufangen – also definieren wir PAGE. Alter Text verschwindet mit Cursor oben links.

SPACE \ . .” xx “ : ; CR PAGE - und damit sind wir bei 9 Worten

Wir muessen den aktuellen Status der LED irgendwo abspeichern, und der Wert muss veraenderbar sein und wir nennen sie **VARIABLEs** und koennen Daten dort speichern und auch auslesen. Fuer die 16 LEDS koennte man 16 davon definieren. Zum Speichern verwenden wir ein Ausrufezeichen “!” und zum Lesen – was ist denn dort, also “@”.

SPACE \ . .” xx “ : ; CR PAGE VARIABLE ! @ - und sind bei 12 Worten – alles was wir fuer unser Beispiel brauchen.

Und damit sind wir am Ende von Teil 1a. Teil 1b zeigt den entsprechenden Code. Mehr Erklaerungen dann in Teil 2.

`End of Part 1 – more at` https://wiki.forth-ev.de/doku.php/projects:a-start-with-forth:start0

```
\ SELBST ERFINDEN STATT ERKLAEREN eine kleine Anwendung auf 2 Seiten - ExMark Nov2016
\ INCLUDE C:\VFXTESTAPP\TESTAPPv10.f

\ Zuerst einmal wollen wir zwei Zeilen auf den Bildschirm bekommen, und dann PW setzen.
\ PW_T3_T2_T1  O3_O2_O1_O0  I3_I2_I1_I0  A3_A2_A1_A0 set XXh/l
\ 1   0  0  0   0  0  0  0   1  1  1  1   0  0  0  0

\ Alle verwendeten Forth-Worte in derselben Reihenfolge wie vorher erklaert:
\ SPACE   \   .   ." xx "   :   ;   CR   PAGE   VARIABLE   !   @

\ Um es zu verkuerzen, hier beschraenkt auf den Anfangsteil, ohne die anderen 15 LEDs.

\ Definiere die Worte, die den Header auf den Bildschirm schreiben. Und einen Status 0.
: TEXT0   ." ERFINDEN STATT ERKLAEREN, eine kleine Anwendung v4    ExMark Oct2016 " CR ;
: TEXT1   ." PW    PWH um auf 1 und PWL um auf 0 zu setzen " CR ;
: TEXT2   ." 0   " CR ;

\ Definiere die Variable PW (spaeter Pulse Width), die den Status der PW-LED speichert.
VARIABLE PW        \ die Variable PW definieren

\ Jetzt die 2 Worte, die den Status der PW-LED in der Variablen 0(Low), 1(High) setzen.
: PWH   1 PW ! ; \ definiere ein Wort PWH, und den Wert 1, Variable ist PW, dort hinein
: PWL   0 PW ! ; \ definiere ein Wort PWL, und den Wert 0, Variable ist PW, dort hinein

\ Und jetzt die Worte, um den augenblicklichen Wert auf dem Bildschirm anzuzeigen.
\ Zuerst ein Wort DISBIT, um PW und den Wert der Variablen auszugeben.
: DISBIT   PW @ . SPACE   ;   \ -Definiere -Name DISBIT -PW Inhalt -holen -ausgeben -SPACE

\ Als Uebung den Namen DISBIT auch als BITPW definieren - wurde dann nicht verwendet.
: BITPW    DISBIT ;

\ Und jetzt diesen Text und den Inhalt von PW an den Bildschirm schicken.
: Display    CR TEXT1 DISBIT ;   \ DISPLAY schickt CR, Zeile TEXT1, sowie PW raus

\ PWH und PWL aendert den Status der LED in der Variablen, DISPLAY zeigt ihn an.

\ Zum Abschluss noch ein neues Wort MS (Millisekunden-Verzoegerung).
\ Der Code xx MS implemeniert eine Verzoegerung von xx ms, vor weiterer Codeausfuehrung
\ Zum Beispiel 1000 MS verzoegert 1000 ms (dezimal).
: ONOFF   PWH DISBIT 1000 MS PWL DISBIT ;   \ ONOFF setzt LED HIGH nach 1000ms auf LOW

: Hello    ." HELLO FORTH WORLD   " ;  \ … noch der in Sprachen oft erste Programmcode.
\ Hello <cr> zeigt dann HELLO FORTH WORLD an.

\ Kurzbeschreibung der hier verwendeten Forth-Worte:
\ SPACE     Ausdruck einer Lehrstelle auf dem Bildschirm
\ \         Das Trennungswort \ sagt dem PC, alles bis zum Ende der Zeile ist Kommentar
\ .         Schicke einen Wert zum Bildschirm
\ ." xx "   Schicke den ASCII-Text zwischen Dot-Quote und Quote an den Bildschirm
\ :         Starte eine neue Wortdefinition, zum Beispiel wie in : Hallo ." Hello " ;
\ ;         Diese neue Wortdefinition wird mit dem Semicolon ; abgeschlossen.
\ CR        Schicke per Programm <cr> zum Bildschirm - eine Zeile runter, Cursor links
\ PAGE      Loesche alle Information auf dem Bildschirm, starte mit einer leeren Seite
\ VARIABLE  Definiere eine neue Variable, hier VARIABLE PW - nur niedrigstes Bit genutzt
\ !         Speichere den Wert xx in der Variablen:  xx PW !
\ @         Lies den Inhalt der Variable in Memory an und zeige Inhalt an mit   PW @ .
\ MS        MS definiert Verzoegerung  xxx ms, dann geht es weiter, z.B. 1000ms = 1sec

\ Ende von Teils 1a. Erste Erfolge mit LED-Kontrolle, HELLO WORLD und Verzoegerung.
\ Teil 2 erklaert wie es intern ablaeuft, und ein kleiner Debugger wird entwickelt.
```

End of Part 1a - more at https://wiki.forth-ev.de/doku.php/projects:a-start-with-forth:start0

Eine Sprache selbst erfinden – anstatt sie zu erklaeren - Teil 2 v16

Im Teil 1 hatten wir die Grundlagen unserer Sprache “erfunden”. Hier in Teil 2 wird es komplexer und wir bauen einen kleinen Debugger auf, damit wir sehen was wo passiert.

Anstatt komplizierter Erklaerungen, auch hier ein bisschen herumgeschaut um zu sehen, wie andere Ablaeufe funktionieren: Einmal wie es im Buero funktioniert, oder wie Autos am Fliessband gebaut wurden. Wir werden den Ablauf aehnlich aufbauen, und dann die Hardware dafuer aufbauen. Als Nebeneffekt ergibt es sich dann, dass bei mathematischen Formeln zum Beispiel die Klammern verschwinden, aber Sequenzen umsortiert werden muessen um einen Fertigungsprozess vom Fliessband zu ermoeglichen.

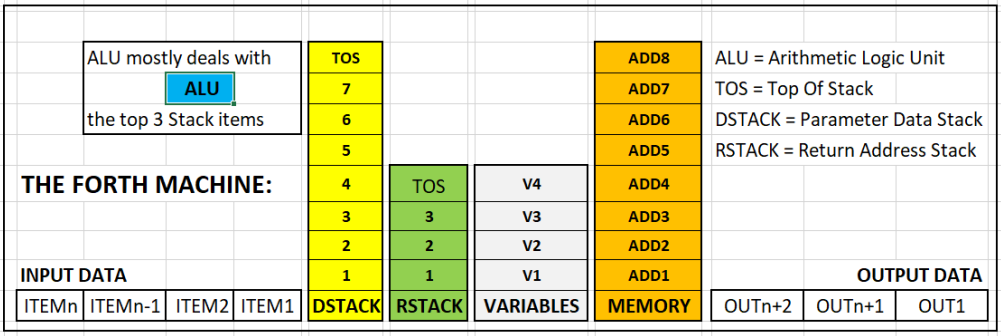
Als erstes Beispiel: Man sitzt am Schreibtisch, die **Daten und Anweisungen kommen von links** als kleine Kartons. Der Inhalt muss entweder gleich verarbeitet werden, oder irgendwo abgelegt werden fuer spaetere Bearbeitung. Also legen wir diese Dinge in der Warteschlange auf einen Stapel – und nennen ihn **DSTACK**. Neues kommt einfach oben drauf. Je besser diese vorsortiert sind bevor sie ankommen, desto weniger sind es. Manchmal ruft der Chef an, eine Unterbrechung (Interrupt), oder man muss etwas anderes weiterverarbeiten. Was man in der Hand hat, landet dann auf einem weiteren Stapel, wo man dies dann wieder abholt wenn man an der alten Stelle weitermacht – der **RSTACK** – Return Stack. Manchmal muss man sich einfach etwas fuer direkten Zugriff merken, legt sie nebeneinander auf den Tisch und nennt sie Variable - **VARIABLE**. Sie arbeiten wie benannte Speicherzellen - Plaetze auf dem Tisch.

Man kann auch **Speicherbereiche** in **MEMORY** benennen mit verschiedener Laenge – aber das kommt spaeter.

Die **ALU** ist mit den 3 obersten DSTACK Plaetzen verbunden, und die Ergebnisse landen dort auch wieder.

Nach der internen Verarbeitung werden die Ergebnisse eventuell ueber den Ausgang geschickt, die Ausgangsdaten.

Und genau diese FORTH-MASCHINE, die wir gerade beschrieben haben, werden wir in Zukunft verwenden.



Wir haben diese Maschine schon ohne Erklaerung in Teil1 verwendet. Jetzt koennen wir genauer darauf eingehen. Und weil es jetzt komplizierter wird, muessen wir uns etwas ausdenken um anzuzeigen, was innendrin passiert.

Ein kleiner Monitor/Debugger wird uns erst mal einige Details auf dem Bildschirm zeigen, mehr wenn notwendig: Einige Variablen – einige obere Positionen auf den Return Stack – und einen Teil des oberen Data Stacks.

Zuerst mal definieren was wir mit dem Monitor anzeigen wollen, dann wissen wir, welche Worte noch noetig sind:

```
: DebugDisplay .” V1_V2_V3_V4___R0_R1_R2___D0_D1_D2_D3_D4_D5_D6_D7 “ \ Stack-Positionen relativ nicht absolut!
```

Auf diese Weise kann man 4 Variable, die Top 3 RSTACK und die 8 Top DSTACK Positionen anzeigen.

How to display the 4 Variables we know already: `V1 @ . V2 @ . V3@ . V4 @ .` and so a first word is

```
: VARS PW @ . space T3 @ . space T2 @ . space T1 @ . space ;
```

Mit ein bisschen Daten hin- und herschaufeln kann man anschauen, was auf den 3 hoechsten RSTACK Positionen liegt. Aber dafuer brauchen wir zwei neue Worte:

R> schiebt die Daten Stueck fuer Stueck vom RSTACK auf den DSTACK und kann dann angezeigt werden.
>R schiebt die Daten in die umgekehrte Richtung – vom Data Stack auf den Return Stack.

>R >R >R wird damit die 3 obersten relativen Positionen R2, dann R1 und R0 auf den Data Stack verschieben; und diese koennen dann angezeigt werden. Ein neues Wort das dies kombiniert waere als erster Versuch:

```
: RSTACK R> R> R> . space . space . space . ; \ aber diese Daten muessen zurueck zum RSTACK!
```

Nun zum DSTACK. Auch hier ein bisschen Datenschaufeln und es geht. Es gibt sicher bessere Methoden – aber dies kann jeder verstehen. 8 Stapeleintraege ruebergeschoben von DSTACK nach RSTACK mit >R >R >R >R >R >R >R >R. Die Reihenfolge ist D7 D6 D5 D4 D3 D2 D1 D0, auf dem RSTACK. Dies kann jetzt wieder zurueckgeschoben und gleich ausgedruckt werden. Das Wort . entfernt aber leider den DStack-Eintrag wie es auch sein sollte. Dieses Problem loesen wir, indem wir den DStack-Eintrag mit **DUP** duplizieren bevor wir ihn mit . anzeigen. Das erste Wortschiebt die Daten rueber, das 2. Wort holt 4 zurueck und zeigt sie an, das 3. Wort fuer die zweiten 4, und das 4. Kombiniert die 3.

```
: DSTACKtoRSTACK >R >R >R >R >R >R >R >R ;                                  \ 1
: DtRbackDUP1_4. R> DUP . space R> DUP . space R> DUP . space R> DUP . space ; \ 2
: DtRbackDUP5_8. R> DUP . space R> DUP . space R> DUP . space R> DUP .   ;    \ 3
: DSTACK DSTACKtoRSTACK DtRbackDUP1_4. DtRbackDUP5_8. ;                     \ 4
```

Und das wieder in ?? kombiniert, schickt den Status unserer Forth-Maschine auf den Bildschirm:

```
: ?? VARS RSTACK DSTACK ;
```

ABER, leider muessen wir ja zugeben, dass wir hier in eine laufende Maschine hineingreifen. Wir muessen sicherstellen, dass der normale Programmablauf und unser Debugger nicht kollidieren. Von daher kann es je nach Forth-Inplementierung Probleme geben. Zur Sicherheit sollten ALLE DEBUGGER AKTIVITAETEN in einem Forth-Wort ohne Unterworte codiert warden – also zwischen **:** und **:** . Das ist zwar schlechter lesbar – aber funktioniert.

Hier noch einmal der komplette Code – auch von Teil 1 – alles auf einer Seite und jetzt alle 16 LEDs kontrollierbar.

```
\ SELBST ERFINDEN STATT ERKLAEREN eine kleine Anwendung auf 2 Seiten - ExMark Nov2016
\ INCLUDE C:\VFXTESTAPP\TESTAPPv10.f

\ Zeige diese zwei Zeilen an
\ PW_T3_T2_T1  O3_O2_O1_O0  I3_I2_I1_I0  A3_A2_A1_A0 set   XXh
\ 1  0  0  0   0  0  0  0   1  1  1  1   0  0  0  0  reset XXl

\ Alle verwendeten Forth-Worte von Teil 1 und die paar aus Teil 2:
\ SPACE  \   .   ." xx "  :   ;   CR   PAGE   VARIABLE   !   @   ( >r   r>   DUP )

\ Die 2 Worte um die Textzeilen anzuzeigen
: TEXT0 ." INVENT A LANGUAGE, do a samll application v10  ExMark Oct2016 " CR ;
: TEXT1 ." PW_T3_T2_T1  O3_O2_O1_O0  I3_I2_I1_I0  A3_A2_A1_A0 XXh / XXl "  CR ;

\ 16 Variable definieren, eine per LED der Einfachheit halber, 1 Variable reicht eigentlich.
VARIABLE PW      VARIABLE T3      VARIABLE T2      VARIABLE T1
VARIABLE O3      VARIABLE O2      VARIABLE O1      VARIABLE O0
VARIABLE I3      VARIABLE I2      VARIABLE I1      VARIABLE I0
VARIABLE A3      VARIABLE A2      VARIABLE A1      VARIABLE A0

\ Und die 16 Wort-Paare, um alle 16 LEDs separat ein- und auszuschalten – High Low zu setzen.
: PWH 1 PW ! ;   : T3H 1 T3 ! ;    : T2H 1 T2 ! ;    : T1H 1 T1 ! ;
: PWL 0 PW ! ;   : T3L 0 T3 ! ;    : T2L 0 T2 ! ;    : T1L 0 T1 ! ;
: O3H 1 O3 ! ;   : O2H 1 O2 ! ;    : O1H 1 O1 ! ;    : O0H 1 O0 ! ;
: O3L 0 O3 ! ;   : O2L 0 O2 ! ;    : O1L 0 O1 ! ;    : O0L 0 O0 ! ;
: I3H 1 I3 ! ;   : I2H 1 I2 ! ;    : I1H 1 I1 ! ;    : I0H 1 I0 ! ;
: I3L 0 I3 ! ;   : I2L 0 I2 ! ;    : I1L 0 I1 ! ;    : I0L 0 I0 ! ;
: A3H 1 A3 ! ;   : A2H 1 A2 ! ;    : A1H 1 A1 ! ;    : A0H 1 A0 ! ;
: A3L 0 A3 ! ;   : A2L 0 A2 ! ;    : A1L 0 A1 ! ;    : A0L 0 A0 ! ;

\ Define a word that displays the 16 variables as shown above

\ This needs the spaces for better readability
: 2SP Space Space ;

: Nibble1  PW @ . SPACE  T3 @ . SPACE  T2 @ . SPACE  T1 @ . ;
: Nibble2  O3 @ . SPACE  O2 @ . SPACE  O1 @ . SPACE  O0 @ . ;
: Nibble3  I3 @ . SPACE  I2 @ . SPACE  I1 @ . SPACE  I0 @ . ;
: Nibble4  A3 @ . SPACE  A2 @ . SPACE  A1 @ . SPACE  A0 @ . ;

\ And all 16 bits plus the Spaces in between
: 4NIBBLES  Nibble1 2SP  NIBBLE2 2SP  NIBBLE3 2SP  NIBBLE4 ;
\ And both lines now
: Display  CR TEXT1 4NIBBLES ;

: Hello ." HELLO FORTH WORLD " ;   \ added to show the usual first program

\ Die Worte:  SPACE   \   .   ." xx   "   :   ;   CR   PAGE   VARIABLE   !   @   >R   R>   DUP

\ Teil 2: Der kleine Monitor/Debugger um die Forth-Maschine anzuzeigen V1.V4, RS0..RS2, DS0..7
: v-r-d  ." V1_V2_V3_V4    R0_R1_R2   D0_D1_D2_D3_D4_D5_D6_D7 " CR ;
: vars  PW @ . space T3 @ . space  T2 @ . space  T1 @ . space space space ;   \ Variable anzeigen
: RSTACK  R>  R>  R>  . space  . space . space space space ;                   \ RStack   anzeigen
: DSTACKtoRSTACK  >R >R >R >R >R >R >R >R ;              \ Die 8 DSTACK-WErte auf RStack schieben
: DtRbackDUP1_4.  R> DUP . space  R> DUP . space  R> DUP . space  R> DUP .  space  ; \ 4 anzeigen
: DtRbackDUP5_8.  R> DUP . space  R> DUP . space  R> DUP . space  R> DUP .  space  ; \ 4 anzeigen
: DSTACK  DSTACKtoRSTACK  DtRbackDUP1_4. DtRbackDUP5_8. ;                \ die 3 Worte kombiniert
: ??  VARS  RSTACK  DSTACK  ;                                             \ und alle 3 kombiniert
: RSTACKBASE 0 >R ;          \ Nach dem Start eine 0 auf den RStack – zeigt die unterste Position
: RSTACK12  1 >R 2 >R ;      \ und eine 1 und eine 2
: DSTACKBASE 0 ;             \ Nach dem Start eine 0 auf DStack, um den Startpunkt zu markieren
: DSTACK17 1 2 3 4 5 6 7 ;   \ und darueber 1 2 3 4 5 6 7

\ Der endgueltige Monitor/Debugger; er hat alle diese Worte als eine lange Zeile zwischen : und ;
\ um korrekt mit VFXTESTAPP.exe zu funktionieren – andere Forths moegen anderes Verhalten zeigen.
: ?????  3 spaces PW @ . T3 @ . T2 @ . T1 @ . 4 spaces R>  R>  R> DUP >R . Space DUP >R . space
DUP >R . 4 SPACES  >R >R >R >R >R >R >R >R R> DUP . R> DUP . R> DUP . R> DUP . R> DUP . R> DUP .
R> DUP . R> DUP . ;
\ ????? kann einmal zum interaktiven Testen verwendet werden, oder auch eingebunden in Programme.
```