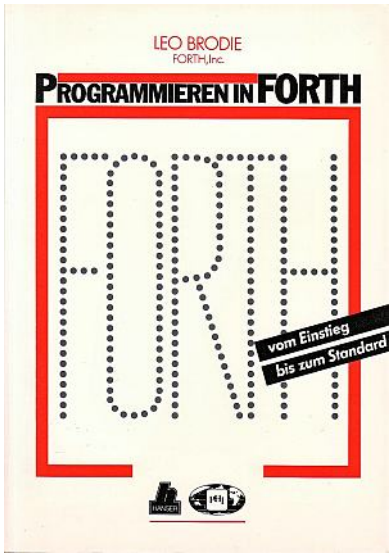
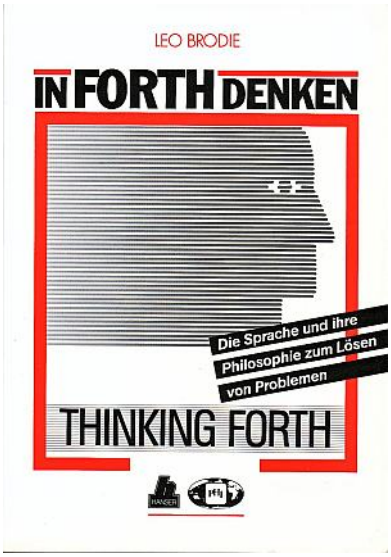
Der erste Außeneinsatz

mental ist nun in einer startbaren Alphaversion angekommen und wird damit für erste Experimente in die freie Wildbahn entlassen, womit eine Minidokumentation fällig wird.
Verfügbar ist dieser "Erste-Schritte"-Text, eine umfangreiche Referenztabelle der aktuell implementierten Worte in **mental** und ein Text mit Überlegungen und Einsichten aus dem "Maschinenraum" (1 - Der Weltraum, unendliche Weite) des Systems. Beide Texte beziehen sich auf die Alphaversion, können also in der weiteren Entwicklung von der Realität überholt werden. Zudem gibt es hier noch keine systematische Einweisung, sondern vorerst nur einige nutzbare Beispiele, die den Weg für den Neugierigen ebnen sollen. Es wird also noch nicht genau erklärt, wie diese forthähnliche Sprache im Detail funktioniert, sondern nur auf Besonderheiten und Abweichungen eingegangen.

Empfehlenswerte Literatur: Für einen systematischen Einstieg lohnen sich die verfügbaren Bücher zum Thema, allsamt leider nur noch in gebrauchter Form zu bekommen:





1. Brodie, Leo: Programmieren in Forth.
2. Brodie, Leo: In FORTH denken.

Beide Bücher sind in der englischen Version auch als eBook im Internet zu finden. Ersteres Buch bezieht sich auf den Forth-79 Standard, was aber eh nicht so relevant ist, da **mental** ganz anders ist... ;) Nein, im Ernst: Es gibt natürlich gravierende Unterschiede im Detail, aber die grundlegenden Konzepte und Ideen gleichen sich. Für Einsteiger ist das Buch "Programmieren in Forth." sehr empfehlenswert. Hier wird in lockerer, einprägsamer und witziger Form alles erklärt, was zu einem Forth gehört.

Aktueller Stand der Entwicklung: Bisheriges Ziel war es, ein forthähnliches System für den Hive zu entwickeln, welches entweder TriOS ersetzt, oder als Programmiersprache unter TriOS gestartet werden kann. Mit der Alphaversion wollte ich folgenden Stand erreichen:

- mCore und Grundwortschatz
- Compiler

- Interpreter
- Quelltexteditor
- Quelltextverwaltung
- TriOS-Loader

Diese Funktionen sind die Grundlage, um autark und ohne Host mit dem Hive zu arbeiten. Ich gehe hier von der Installation mit TriOS aus, da ich denke, dass das wohl in den meisten Fällen so genutzt wird. Da das mental file system (MFS ;) einige Besonderheiten hat, ist es sinnvoll, die Installation auf einer leeren SD-Card unabhängig von anderen Dateien durchzuführen. Wer nach der folgenden Anleitung vorgeht, kann so erstmal problemlos loslegen, ohne andere Dateien und Daten zu beschädigen. Zum MFS gibt es am Schluß noch einige Hinweise.

Installation

Das Archiv von mental enthält folgende Verzeichnisse:

bin	-	Compilierte Dateien für die Installation auf SD-Card unter TriOS
dok	-	Dokumentation
lib	-	Bibliotheken
rom	-	Code für die drei Propellerchips
tapes	-	Containerdateien für mental-Quelltexte und Daten

`mental` kann in zwei Varianten auf dem Hive installiert werden:

1. Installation unter TriOS: Hierbei wird `mental` auf einer normalen SD-Card zusammen mit TriOS installiert. Zwischen TriOS und `mental` kann durch Kommandos gewechselt werden, aber es gibt durc das Dateisystem von mental auch Einschränkungen.
2. Stand-alone: `mental` benötigt kein Betriebsystem wie TriOS, sondern enthält selbst alle wichtigen Funktionen um als Betriebssystem genutzt zu werden. Durch diese Installationsart startet `mental` sehr schnell, da es direkt in den ROMs gespeichert wird. Nachteil: Programme, welche auf TriOS basieren, können momentan noch nicht von dem System gestartet werden. Wer also auf dem Bordcomputer den StarTracker nutzen möchte, sollte `mental` unter TriOS installieren.

Variante 1: Installation von mental zusammen mit TriOS:

1. SD-Card formatieren! **WICHTIG:** Die Karte muss bei der Installation LEER sein!
2. mental installieren: Kopiere alle mental-Dateien aus dem Verzeichnis mental\bin\sdcard auf die SD-Card
3. TriOS installieren: Kopiere alle TriOS-Dateien aus dem Verzeichnis trios\bin\sdcard auf die SD-Card

Variante 2: Stand-alone-Installation von mental ohne TriOS:

1. SD-Card formatieren! **WICHTIG:** Die Karte muss bei der Installation LEER sein!
2. Kopiere die Datei "sys" und "usr" sowie tap0..7 aus dem Verzeichnis mental\bin\sdcard auf

die SD-Card.

3. Flash \rom\admflash.spin --> Administra
4. Flash \rom\belflash.spin --> Bellatrix
5. Flash \rom\regflash.spin --> Regnatix

Nun kann die SD-Card eingelegt und der Hive gestartet werden. mental startet in dieser Version sehr schnell direkt aus den drei ROM-Speichern.

Energie! - mental starten

Bei einer Installation mit TriOS landet man nach dem Systemstart ganz normal in der Kommandozeile Regime. Zeigt man das Verzeichnis an, sollte es in wie folgt aussehen:

```
 ok
~ dir wh
o Datenträger : mental
  usr       sys         tap0        tap1
  tap2      tap3        tap4        tap5
  tap6      tap7        adm         bel
  reg       reg.sys     ▶SYSTEM     ▶AYS
  ▶HSS      ▶SFX        ▶SID        ▶TBOX-1
  ▶TBOX-2   ▶TRIBORG    ▶WAV

o Anzahl der Dateien : 23 ok
```

Die Dateien **sys, usr** und **tap0..7** sind sogenannte "Tapes" - Containerdateien, in welchen **mental** Quelltextscreens und Daten speichern kann. Im Prinzip kann man später noch weiter Tape-Dateien mit anderem Namen anlegen, zum Beispiel um ein bestimmtes Projekt darin zu speichern. Ebenso können diese Dateien so auch problemlos kopiert und weitergegeben werden, aber dazu später mehr im Abschnitt zum MFS.

Zusätzlich ist das typische SYSTEM-Verzeichnis von TriOS sichtbar und die Datei reg.sys, welche ja die Kommandozeile Regime enthält.

Bei den drei Dateien **adm**, **bel** und **reg** handelt es sich um den Chipcode von mental für alle drei Mikrocontroller. Im Systemverzeichnis befindet sich auch die Datei **m.bin,** der mental-Loader - ein ganz normales TriOS-Programm, welches dem Gerät quasi eine "Gehirnwäsche" verpasst und **mental** als völlig neues System startet.

Also los, starten wir einfach mental durch die Eingabe des Kommandos **m:**

```
~ m

Wir sind Borg, Widerstand ist zwecklos.
Assimilation wird gestartet...

Check ADM-Code : ok
Check BEL-Code : ok
Check REG-Code : ok
Check USR-Tape : ok
Check SYS-Tape : ok

System wird nun gestartet...
```

Diese Meldungen sind im Normalfall kaum sichtbar, sofern alles korrekt installiert ist. Fehlt eine Datei, so stoppt der Loader und gibt entsprechende Hinweise zur Fehlerursache. Folgend wird als erstes der Bellatrix-Code ausgetausch, wodurch der Bildschirm gelöscht wird und man kann das weitere Laden der Chipcodes für Administra und Regnatix verfolgen.

Letztlich sollte folgender Screen sichtbar sein:

```
willkommen zu mental

ok









[execute]
```

Am unteren Rand findet sich eine Eingabezeile mit Cursor. Der Status [execute] weist darauf hin, dass sich das System im Interpretermodus befindet, alle Eingaben werden also sofort interpretiert und nach Betätigung der Eingabetaste ausgeführt. Probieren wir es mit einer ersten Eingabe: Mit der Eingabe **tapes** sollte nun folgendes auf dem Screen erscheinen:

Schauen wir uns das in Ruhe an. Der eingegebene Befehl wird im oberen Logbereich noch einmal ausgegeben, darauf folgt ein rotes Dreieck. Dieses Dreieck trennt die Eingabe optisch von der Ausgabe des Systems. Im Fall des Wortes **tapes** werden alle Tapes auf der SD-Card aufgelistet. Später kann mit einem beliebigen Tape gearbeitet werden, nach dem Systemstart ist automatisch das Tape **usr** ausgewählt, welches quasi der Benutzerbereich darstellt.

Weiter im Text: teste den Befehl **words**. Auf dem Bildschirm werden nun alle Worte aufgelistet, die sich im Wörterbuch befinden.

Die Schiffsdatenbank

`mental` besitzt wie alle Forthsysteme einen viel größeren Bezug zur natürlichen Sprache, als viele andere Programmiersprachen. Zumindest bietet es das Potential, sofern man sich auf das Konzept einläßt. In Forth kann man natürlich auch wie in C programmieren, was aber in etwa dem Versuch gleicht, mit einer Katze einen Nagel in die Wand zu schagen - funktioniert, ist aber eine mächtige Sauerei. Umdenken ist also angesagt.

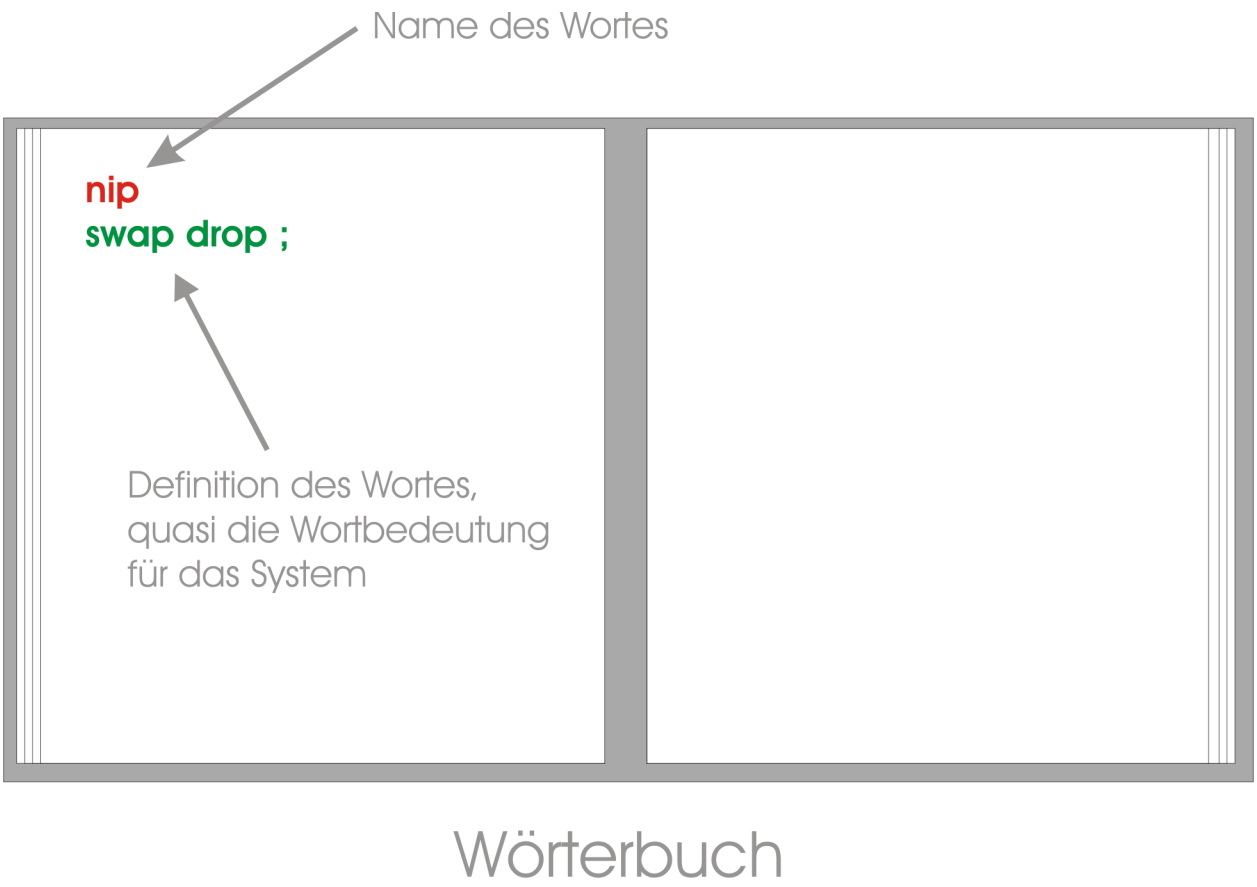
Eines der wichtigsten Konzepte der Programmiersprache Forth ist das Wörterbuch, eine einfache interaktiv erweiterbare Datenbank mit allen Befehlen des Systems. Einen Befehl in der Form eines Unterprogramms bezeichnet man entsprechend als "Wort". Wie in einem realen Wörterbuch sind nun alle (!) verfügbaren Wörter in diesem Wörterbuch eingetragen. So findet man hier alle Bestandteile des Compilers, des Interpreters als kleine interaktiv nutzbare Module. Selbst die

Befehle zur Anzeige und Verwaltung des Wörterbuches selbst sind Wörter in der Datenbank. Wenn wir also alle Wörter des Buches anzeigen wollen, können wir das ganz intuitiv mit der Eingabe von **words** tun. Am besten gleich ausprobieren, es sollte etwa folgende Ausgabe erscheinen:

```
willkommen zu mental

ok
words ▶ sys edsys hallo [usr] br .scr .line format blank index+ index0
index .index page index# bye reboot [sys] last reset tapes use (use) ?cerr cold
m --> load s> >s (load) eos? perform ed0 ed- ed+ scrmax scrnr edwr edrd list
pos? scr! scr@ fill write read tab ?mount unmount mount ed x! x@ clkfreq wait
cnt again until bein leave +loop loop do forget empty if else the for i next
digits free nfa>pfa align find =name =str bin hex dec base ?error error abort .s
ds compile ?s ?p ?i ?csp csp long word data create ; ;data string sliteral
nliteral reduce lit6 lit16 lit32 new strcopy smove , allot +! interpret token
function atoken btoken nop .str str1 str0 tok words cls cr spaces space link 1-
1+ .name . inkey key emit b@s b@l b!l <8 b@ b! a!s a@s a@l a!l a@ a! execute
exit c! c@ w! w@ ! @ negate abs / /mod 2dup sign um/mod * um* - + <rot rot>
shift> <shift not xor or andn and 0= <> > = < nip swap r> >r rot pick dup drop
here dp  ok




[execute]
```

words ist manchmal praktisch, aber für den Einsteiger sicherlich schrecklich, da alle Befehle unstrukturiert sichtbar sind. Ich glaube Charles Moore selbst (der Schöpfer von Forth) hat mal gesagt, dass man dieses Wort besser verbieten sollte... Da **mental** aber sehr einfach und klein ist, bleibt die Übersicht des Wörterbuches dennoch im Rahmen.

Im Prinzip kann man das Konzept des Wörterbuches ganz wörtlich nehmen und sich auch ein entsprechendes Bild davon machen: Ein Buch, welches auf jeder Seite am oberen Rand in roter Farbe ein Wort stehen hat und darunter ein Erklärung der Bedeutung dieses Wortes. In Forth nun ist die Bedeutung aber wieder in Forth selbst geschrieben. Die Bedeutung eines Wortes in Forth wird in der Sprache Forth, also mit anderen schon vorhandenen Worten erklärt. Das Besondere an diesem Wörterbuch ist seine strenge chronologische Struktur: Wird auf Seite 10 zum Beispiel die Bedeutung des Wortes nip definiert, so kann das nur mit Wörtern der Wörter auf den vorherigen neun Seiten geschehen. Dieser Umstand ist auch ganz logisch und natürlich, denn ich kann ein neues Ding nur mit schon bekannten Wörtern beschreiben.



Wörterbuch

Geben wir also in der Eingabezeile **nip** ein, so sucht Forth im Wörterbuch bis es die Seite mit der entsprechenden Definition **swap drop ;** findet. Das Zeichen **;** ist übrigens ebenfalls ein Wort. In **mental** können Wortnamen 15 Zeichen lang sein und alle Zeichen enthalten. Die einzige Syntaxregel besteht in der Regel Wörter mit Leerzeichen zu trennen und die Bedeutung der verschiedenen Farben, zu denen wir später noch kommen.

Die Sequenz **swap drop ;** ist also eine Abfolge von drei Befehlen, welche quasi die Bedeutung desWortes nip definieren - geben wir **nip** ein, so führt **mental** diese Sequenz aus, realisiert also die Bedeutung des Wortes. In der Realität stehen diese Sequenzen aber nicht in Textform im Wörterbuch, sondern sie werden nach der Eingabe vom Compiler in einen Maschinencode übersetzt und als ausführbarer Code dort gespeichert.

Aber damit nicht genug: Wir können zu jedem Zeitpunkt dieses Wörterbuch erweitern. Probieren wir es aus! Wem **words** nicht gefällt, kann dafür einen anderen Namen verwenden, indem ein neuer Eintrag im Wörterbuch erzeugt wird. Einen passenden neuen Eintrag erhalten wir mit folgender Eingabe:

```
ok

[execute]
wörter words ;
```

Die Farben werden dabei mit den Funktionstasten gewählt.

Rot F2 - Name des neuen Wortes im Wörterbuch.

Grün F3 - Definition der bedeutung des Wortes.

Die Eingabe erfolgt also folgendermaßen:

F2 wörter F3 words ;

Ist eine Farbe falsch gesetzt, so kann das später problemlos korrigiert werden: Einfach den Cursor auf ein beliebiges Zeichen des entsprechenden Wortes setzen und eine Funktionstaste betätigen. Mit dieser Methode können alle Farben durchgespielt werden, bis alles korrekt ist. Die Tabulator-Taste lässt den Cursor von Wort zu Wort springen. Erst bei der Betätigung der Eingabetaste wird unsere neue Definition übernommen. Ist alles korrekt, sollte folgendes sichtbar sein:

```
ok
wörter words ;  ok

[execute]
```

Nun wollen wir unsere Erweiterung des Wörterbuches überprüfen, indem wir **wörter** eingeben. Der Inhalt unseres erweiterten Buches wird angezeigt, **mental** versteht also nun was wir meinen, wenn wir dieses neue Wort benutzen.

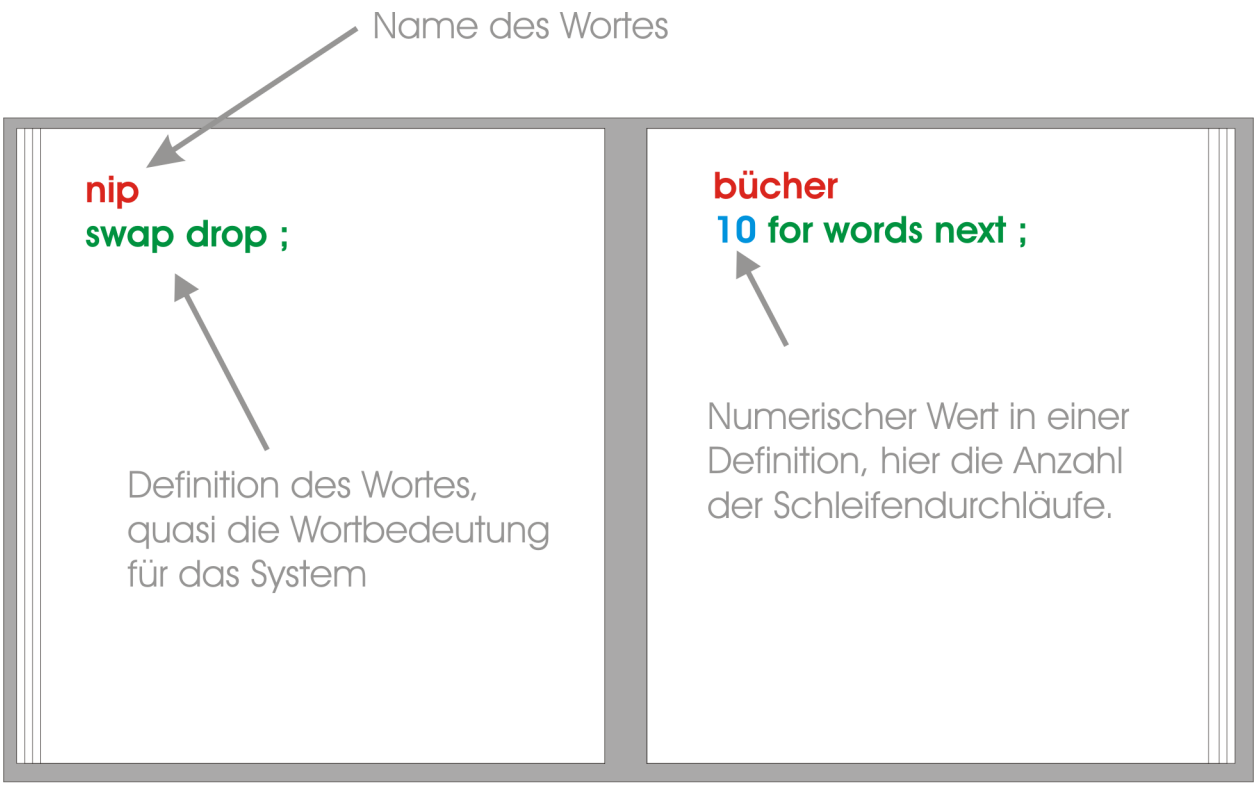
Mit diesem Konzept können wir beliebig komplexe Bedeutungsinhalte definieren. Dazu sind bei komplizierten Problemen vielleicht viele neue Wörter nötig, aber letztlich gibt es ein letztes Wort **run, pacman** oder **windows-8**, welches man eingeben kann und die ganze Sache läuft los.

Und an diesem Punkt wird der Unterschied im Umgang mit Sprache bei Forth sichtbar: In Forth programmiert man nicht, sondern Forth ist selber das Programm welches erweitert wird, bis es tut was wir wollen. Forth ist dabei immer präsent, bleibt immer Teil der Anwendung. Natürlich gibt es dann später auch eine Möglichkeit, ein Wort automatisch zu starten, damit der Anwender nichts mehr von Forth sieht und die fertige Anwendung wie ein normales Programm starten kann.

Noch einige Informationen zur Eingabezeile: Sollte eine Eingabe länger sein als die Eingabezeile, so kann die Definition auch über mehrere Eingaben erfolgen. Bitte folgendes eingeben:

```
bücher RETURN
10 for words next ; RETURN
```

Mit dieser Eingabe haben wir ein weiteres Wort auf einer neuen Seite im Wörterbuch in seiner Bedeutung definiert.



Wörterbuch

Mit der Farbe Cyan werden übrigens numerische Werte gekennzeichnet, welche bei der Ausführung des Wortes (numerisches Literal) benötigt werden. Cyan wird mit der Taste F5 ausgewählt. Das Wort gibt zehn mal das Wörterbuch aus. Wie man sieht, kann die Definition während der Eingabe auch über mehrere Zeilen erfolgen. Oft wird das nicht nötig sein, da Forth-Definitionen die Tendenz zu vielen kleinen und einzeiligen Wörtern hat und weil diese Definitionen meist auch im Quelltexteditor eingegeben und von dort aus compiliert werden.

Wie werden nun aber Wörter wieder aus dem Wörterbuch gelöscht? Ganz einfach:

```
bücher forget
```

Mit **forget** löscht man alle Wörter ab dem Wort welches als String (Orange = F6) übergeben wurde. Es kann also kein Wort mitten im Wörterbuch gelöscht werden, sondern nur ein ganzer Abschitt von einem Wort aus bis zum Ende des Wörterbuches!

Farben und Bedeutungen

Noch einmal zur Statusanzeige **[execute]** im unteren Bildschirmbereich. **mental** verwendet verschiedene Farben um dem Interpreter zu zeigen, um welche Art Wort es sich handelt. Wie wir schon im letzten Abschnitt gesehen haben, können die verschiedenen Varianten mit den Funktionstasten gewählt werden. Einfach ausprobieren, die Statusanzeige wechselt dabei zu folgenden Werten:

```
F1:   [execute]          Wort ausführen/interpretieren
F2:   [create]           Name eines neuen Wortes
F3:   [compile]          Definition eines Wortes
F4:   [number]           Zahlenwert
F5:   [number-literal]   Zahlenwert in einer neuen Definition
F6:   [string]           Zeichenkette
F7:   [string-literal]   Zeichenkette in einer neuen Definition
F8:   [data]             Name einer Variable
F9:   [remark]           Kommentar
```

Ein sehr einfaches Wort haben wir schon im letzten Abschnitt definiert. Nun wollen wir die anderen Farben und ihre Anwendung kennenlernen. Beschäftigen wir uns als erstes mit Zeichenketten.

Im Interpreter können wir Zeichenketten orange kennzeichnen. Probieren wir es aus:

```
Ich bin eine Zeichenkette! .str
```

Das Wort **.str** ist eine Stringausgabe, also quasi eine Print-Anweisung für eine Zeichenkette. Die Abürzung mit einem Punkt erscheint anfänglich ein wenig kryptisch, da man ja in Forth alle Funktionen gut benennen könnte. Wer mag, kann sich auch wirklich einfach ein Wort

```
printstring .str ;
```

definieren, um das zu tun. Für sehr oft genutzte grundlegende Funktionen wurden aber in der Vergangenheit solche symbolischen Wortnamen benutzt, um den Quelltext kompakt zu halten. Aber zurück zur Verwendung von Zeichenketten. Korrekt eingegeben, passiert folgendes:

```
ok
Ich bin eine Zeichenkette! .str ▶ Ich bin eine Zeichenkette! ok

[execute]
```

Eine Zeichenkette in orange landet also in einem Stringpuffer und wird mit dem Wort **.str** oder **printstring** (<-- Probieren!) ausgegeben. Woher diese beiden Wörter wissen wo sich die Zeichenkette befindet, werden wir im nächsten Abschnitt erfahren. So viel vorweg: es ist ein sehr einfacher, universeller und zugleich mächtiger Mechanismus.

Nun wollen wir uns ein neues Wort mit diesem Wissen erstellen:

```
hallo Hallo Welt! .str ;
```

Erstaunt stellen wir fest, das es zu einer Fehlermeldung kommt:

```
ok
hallo Hallo Welt! .str ; ▶ structure imbalance ok

[execute]
▮
```

Was ist geschehen? Die Farbe Orange kennzeichnet nur Zeichenketten für den Interpreter, nicht für die Verwendung in einer neuen Definition, also zur Laufzeit eines Wortes! Der Interpreter verwendet die Zeichenkette unmittelbar, somit wird sie lediglich in einen passenden Puffer übertragen. Bei einer Wortdefinition muss die Zeichenkette aber quasi "eingefroren" werden, um sie erst bei Aufruf des Wortes verfügbar zu haben. Kurz: in **mental** kennzeichnet die gelbe Farbe Zeichenketten für eine spätere Verwendung. Bildlich gesprochen werden gelbe Zeichenketten mit in des Wörterbuch geschrieben. Unser obiges Beispiel würde nur **hallo .str ;** in das Wörterbuch schreiben, wenn der "Buchschreiber" (Compiler) nicht erkennen würde, dass es dort eine Unregelmäßigkeit gäbe und diese mit einer Fehlermeldung abbrechen würde.

Korrigieren wir unsere Eingabe:

```
hallo Hallo Welt! .str ;
```

Aus Orange wird Gelb, und gelbe EIngaben werden als Stringliteral in eine Definition geschrieben. Damit haben wir das neue Wort **hallo** im Wörterbuch erschaffen (compiliert). Mit dem Befehl **words** kann man das wieder einfach überprüfen, ebenso kann man **hallo** einfach aufrufen:

```
hallo ▶ Hallo Welt! ok
```

Was ist hier passiert? Betrachten wir die Definition:

hallo	Eine neue und leere Seite im Wörterbuch für die Definition des Wortes "hallo" wird aufgeschlagen.
Hallo Welt!	Dieser String wird für die spätere (Laufzeit) Verwendung dort notiert.
.str ;	Ausgabe der Zeichenkette und Ende wird notiert

Wird hallo aufgerufen, geschieht folgendes:

1. Schritt: Suche im Wörterbuch nach **hallo**.
2. Schritt: Dort notierte Zeichenkette lesen und "aktivieren".
3. Schritt: Zeichenkette auf dem Bildschirm ausgeben.
4. Schritt: Wort beenden.

Um den Unterschied noch einmal sichtbar zu machen, folgender Versuch:

```
hallo2 Interpreter! .str Laufzeit! .str ; ▶ Interpreter!  ok
hallo2 ▶ Laufzeit!  ok
```

Wie man sieht, wird der erste String "Interpreter!" sofort vom Interpreter ausgegeben, der Zweie String "Laufzeit!" erst, wenn das Wort benutzt wird.

Neben Zeichenketten gibt es auch für numerische Werte zwei Farben:

```
F4:    [number]          Zahlenwert
F5:    [number-literal]  Zahlenwert in einer neuen Definition
```

Sowohl bei Zeichenketten wie auch bei numerischen Werten habe ich versucht die Farben sinnvoll zu wählen. Der dunklere Farbton (**blau** & **orange**) ist für die Verwendung im Interpreter, die hellere Farbvariaton (**cyan** & **gelb**) ist für die Verwendug zur Laufzeit gedacht, müssen also vom Compiler verarbeitet werden.

Der Stack und Postfix-Notation

Forth basiert im Kern auf dem Konzept einer Stackmaschine mit zwei Stacks. Alle Worte verarbeiten Daten ausschließlich direkt auf dem Datenstack. Zusätzlich ist der Datenstack auch interaktiv in einfachster Weise direkt im Interpreter nutzbar, was unschätzbare Vorteile bei Test und Fehlersuche mit sich bringt. Probieren wir es aus:

```
11 22 33 .s ▶ 11 22 33  ok
. . . ▶ 33 22 11  ok
.s ▶  ok
```

Das Wort **.s** zeigt dabei zerstörungsfrei den aktuellen Inhalt des Datenstacks an, das oberste Stackelement befindet sich dabei ganz rechts und wir finden in der Ausgabe so unsere drei Zahlenwerte wieder. Das Wort Dot (der Punkt) gibt das oberste Stackelement als numerischen Wert auf der Konsole aus.

Worte entnehmen dem Stack Daten und speichern ein mögliches Ergebnis wieder auf dem Stack. So entnehmen die Worte **+ - * /** die beiden obersten Werte vom Stack und legt das Ergebnis der Operation wieder auf dem Stack als oberstes Elemennt ab. Zusammen mit dem Dot-Wort können wir nun **mental** interaktiv als Taschenrechner nutzen:

```
1 2 + . ▶ 3        entspricht 1 + 2
2 3 4 + * . ▶ 14   entspricht (4 + 3) * 2
```

Wie man erkennen kann, führt die direkten Einbindung des Datenstacks in den Interpreter ganz natürlich zur Postfix-Notation: Erst werden die Parameter auf dem Stack gesammelt, dann die Operationen mit den Daten durchgeführt. Warum ist das so praktisch? Definiere folgende Worte:

```
sekunden clkfreq * ;
warten wait ;
```

Mit diesen beiden Worten können wir nun schreiben

```
7 sekunden warten ▶  ok
```

und unser Hive wird eine Pause von exakt sieben Sekunden einlegen. Das ist sehr elegant und eine sehr natürliche Methode um zu programmieren und in dieser Form und Einfachheit in keiner anderen Sprache zu realisierbar. Zudem muss man sich wirklich die Einfachheit dieses Mechanismus vor Augen halten, denn wir bewegen uns hier ja noch ganz dicht über dem "blanken Metall" und so verbraucht diese Eleganz auch kaum Ressourcen. Vorraussetzung ist natürlich eine überlegte Wahl der Wortnamen.

Zum Beispiel: Das Wort **warten** verzögert die Programmausführung für die Anzahl an Prozessorticks, welche als Parameter dem Stack entnommen werden. Das Wort **sekunden** konvertiert Sekunden in Prozessorticks. Jedes Wort kann nun einzeln sofort getestet werden:

```
7 sekunden . ▶ 560000000 ok
```

Allgemein wird das Stackverhalten eines Wortes in einem Stackkommentar dargestellt. Schauen wir uns das an einigen Beispielen an:

```
sekunde     (sec -- ticks)
warten      (ticks -- )
+           (n1 n2 -- n3)
```

Der Stackkommentar zeigt den Stackzustand vor und nach der Ausführung desWortes und ist ein völlig funktionsloser reiner Kommentar nur für den Programmierer. So erwartet unser Wort **sekunde** auf dem Stack als obersten Wert einen numerischen Zeitwert in Sekunden, wandelt diesen in Prozessorticks um und speichert das Ergebnis wieder auf dem Stack. Wird das Wort **sekunde** ausgeführt geschieht folgendes:

```
sekunde     (sec --)
clkfreq     (sec -- sec clkfreq)
*           (sec clkfreq -- ticks)
;           (ticks -- ticks)
```

Eine gute Methode bei komplexeren Wörtern den Überblick auf dem Stack zu behalten. Um bei der Verarbeitung mit den Parametern auf dem Stack zu jonglieren, gibt es einige universelle Worte zur Stackmanipulation:

```
drop        (n -- )
swap        (n1 n2 -- n2 n1)
dup         (n -- n n)
rot         (n1 n2 n3 -- n2 n3 n1)
nip         (n1 n2 -- n2)
```

Neben dem Datenstack gibt es noch einen zweiten Stack: den Returnstack. Hier werden die Rücksprungadressen des aufrufenden Wortes gespeichert. Als Einsteiger sollte man vom Returnstack die Finger lassen, denn hier ist großes Konfliktpotential verborgen. Nur so viel: Mit den Worten **>r** und **r>** können Werte zwischen dem Daten- und Returnstack übertragen werden. Ein solcher Wert darf aber auf dem Returnstack nur temporär gespeichert werden, vor dem Ende des Wortes muss der Returnstack wieder ausgeglichen sein, da sonst der Rücksprung durch die fehlerhafte Rücksprungadresse zu einem Systemabsturz führt.

Sowohl Daten- wie auch Returnstack haben aktuell eine Tiefe von 32 Longs und werden direkt im

schnellen Cog-RAM gehalten. Die Größe kann sich aber im Verlauf der Entwicklung noch verändern.

Kontrollstrukturen

Gleich vorweg ein Beispiel: Eine einfache Schleife, welche den Schleifenindex ausgibt.

```
t1 0 do i . loop next ;
10 t1 ▶ 0 1 2 3 4 5 6 7 8 9 10  ok
```

Möglich sind folgende Konstrukte:

- for ... next
- do ... loop
- do ... +loop
- if ... else ... then
- begin ... until
- begin ... leave ... until
- begin ... again
- begin ... leave ... again

Vielleicht auch interessant, um die Geschwindigkeit mit anderen Sprachen zu vergleichen - eine variable Anzahl von Schleifendurchläufen mit einer einfachen Addition zweier Werte:

```
bench for i i + drop next ;
1000000 bench ▶ ok
```

Das Wort benötigt für 1 Millionen Schleifendurchläufe nur etwa 7 Sekunden! Es hat sich also gelohnt iSpin zu vergessen und einen PASM-Warpkern einzubauen... :)

Variablen und Konstanten

Variablen werden in mental ähnlich wie normale Worte definiert:

```
a word ;data            erzeugt eine 16 bit variable
b long ;data            erzeugt eine 32 bit variable
c 16 allot ;data        erzeugt ein array mit 16 bytes
```

Verwendung der Variablen:

```
a @w 1+ a !w            a = a + 1
b @ 1+ b !              b = b + 1
```

Konstanten sind normale Worte mit einem numerischen Literal. Der Compiler codiert die Literal dabei in drei Größenklassen:

```
KONSTANTE_1 1024 ;              konstante
```

Wert: 0..63 - wird als 6Bit-Literal immediate dem Opcode des Literals eingepflanzt
Wert: 64..65535 - wird als Opcode + 16 Bit Wert compiliert
Wert 65536... - wird als Opcode + 2 x 16 Bit Wert compiliert

```
CON_6b 23 ;             speicherverbrauch: 1 x 16 bit
CON_16b 32000 ;         speicherverbrauch: 2 x 16 bit
CON_32b 1000000 ;       speicherverbrauch: 3 x 16 bit
```

Die Angaben zum Speicherverbrauch beziehen sich im obigen Beispiel allein auf das Literal, der Header inklusive dem Namensfeld ist dabei nicht berücksichtig. Es ist warhrscheinlich sinnvoll, Konstantennamen wie in anderen Sprachen groß zu schreiben, um sie von normalen Worten zu unterscheiden.

Ok, jetzt können wir also unser Wörterbuch erweitern indem wir neue Wörter hinzufügen (compilieren). Jedes Wort können wir sofort testen und als Testwerte Werte und Strings auf dem Stack übergeben. Es ist natürlich nicht der Königsweg, alle Quelltexte immer im Interpreter einzugeben. Wie also speichert man sein Quelltexte und wie kann man sie wieder aufrufen?

Editor und Tapes

Quelltexte werden in **mental** wie üblich in Forth als Quelltextseiten mit 16 Zeilen zu je 64 Zeichen gespeichert - den sogenannten "Screens". Anfänglich erscheint eine solche Begrenzung umständlich und wirklich scheiden sich die Geister an dieser Konvention. Einige Forthversionen wie zum Beispiel auch PropForth verarbeiten deshalb auch normale Endlostexte. Ich persönlich aber finde das Screen-Konzept aus folgenden Gründen sehr gut und praktisch:

1. Es ist mit sehr wenigen Ressourcen zu realisieren. Gerade auf einfachen Systemen mit einem blockorientierten Zugriff auf einen Massenspeicher, ist es extrem einfach zu implementieren.
2. Die Begrenzung in der Form zwingt zum Nachdenken und zur genaueren Plaung, was automatisch zu einer besseren Strukturierung führt. Ein Wort kann maximal den Umfang eines Screens haben.

Gespeichert werden die Screens in **mental** in sogenannten *Tapes* welche über einen symbolischen Tapenamen ausgewählt werden können. Innerhalb der Tapes aber findet ein normaler, sehr einfacher und schneller Blockzugriff statt. Ein Block in **mental** entspricht dabei einem Screen, also 1024 Byte. Folgende Worte sind interessant im Umgang mit Tapes:

- **tapes** Ausgabe einer Liste der auf dem Datenträger verfügbaren Tapes
- **name use** Tape name für die Benutzung vorbereiten

- `unmount` SD-Card entfernen
- `mount` SD-Card einbinden
- `index0` Indexzeilen der Screens in einem Tape ab Screen 0 anzeigen
- `index+` Indexzeilen fortlaufend anzeigen
- `scrnr list` Screen mit der Nummer scrnr auf dem Bildschirm ausgeben

Weitere Worte sind der Referenz zu entnehmen. Ebenso handelt es sich bei dem Editor nicht um einen monolithischen Code, sondern vielmehr um einen Editorbaukasten. Hier eine kurze Übersicht über die vorhandenen Bausteine:

- `ed` Editor mit letztem oder leerem Screen aufrufen
- `ed0` Screen 0 in Editor lesen
- `ed+` Screen + 1 in Editor lade
- `ed-` Screen - 1 in Editor laden
- `nr edrd` Screen nr in Editor laden
- `edwr` Screen im Editor auf Datenträger schreiben

Als Beispiel geben wir einfach das Wort **ed0** ein. Der Editor wird mit folgendem Screen sichtbar:

```
screen nr : 0
[usr] ;
hallo Hallo Welt! .str ;









[remark]                        [F10:CPY|11:INS|12:CLR] SCR:0  L: 15   : 1
```

Was wir hier sehe, ist der Screen 0 des Tapes **usr.** In der Grundinstallation ist dieses Tape automatisch aktiv. Wie wir nachher bei der Besprechung des Systemstarts noch sehen werden, wird dieser Screen bei einem Start auch automatisch geladen.

Neben den Funktionstasten für die Colortags sind im Editor aktuell zusätzlich folgende Funktionen verfügbar:

CTRL-N	Screen löschen
CTRL-A	ASCII-Modus einschalten
CTRL-C	Color-Modus einschalten
CTRL-I	Invers-Modus (praktisch zur Eingabe von Strings)
F10	Zeile in Kopierpuffer übertragen
F11	Kopierpuffer in Zeile einfügen
F12	Zeile löschen
ESC	Editor verlassen
TAB	Springt mit dem Cursor zum nächsten Token in der Zeile

Mit diesen Funktione kann problemlos experimentiert werden, denn die Änderungen werden nicht automatisch auf den Datenträger geschrieben. Der Editor kann jederzeit mit der ESC-Taste verlassen werden, mit **ed** kann der Editor mit seinem letzten Inhalt wieder aufgerufen werden.

Achtung: Änderungen werden nach dem Beenden erst durch das Wort **edwr** auf den Datenträger zurückgeschrieben!

Das lässt sich einfach testen: **ed0** aufrufen, Änderungen durchführen, beenden und wieder **ed0** eingeben. Der geänderte Inhalt wurde nun verworfen und der Screen 0 neu geladen. Erst ein **edwr** nach einer Änderung schreibt den Inhalt auf den Datenträger! Ebenso verwirft **edrd, ed+** und **ed-** Änderungen und laden den nächsten bzw. vorigen Screen in den Editor.

Im übrigen sind diese Editorbausteine, die Screens und auch die Tapes wie alle anderen Worte in `mental` uneingeschränkt in eigenen Programmen nutzbar.

Um mir das Leben ein wenig zu erleichtern, habe ich einen Screenbrowser integriert. Dieser wird mit dem Kürzel `br` aufgerufen. Nach dem Start des Browsers wird Screen 0 des aktuellen Tapes dargestellt. Nun können mit verschiedenen Tasten folgende Funktionen aufgerufen werden:

```
page down       Nächster Screen
page up         Voriger Screen
e               Screen editieren
pos1            Screen 0
r               Reboot
w               Write - Screen auf Datenträger schreiben
esc             Browser verlassen
```

Startvorgang und Tapes

In der Grundinstallation (Alpha-Version) haben wir nur zwei Tapes sofort verfügbar, aber keine Angst, es können beliebige neue Tapes hinzugefügt werden.

Beide verfügbaren Tapes haben in mental eine besondere Funktion. Zur Verdeutlichung einfach folgendes eingeben:

```
sys use ed0
```

Das Wort **use** wählt dabei ein bestimmtes Tape für die Benutzung aus, in unserem Fall das sys-Tape. Auf die Farbe dabei achten: sys muss orange geschrieben werden, da es als Tapename ein String ist, welcher von dem Wort **use** verarbeitet wird. Mit **ed0** wird der Screen 0 dieses Tapes aufgerufen. Im Normalfall befindet sich in dem ersten Screen die Ladeprozedur eines Projektes, welches in einem Tape gespeichert ist.

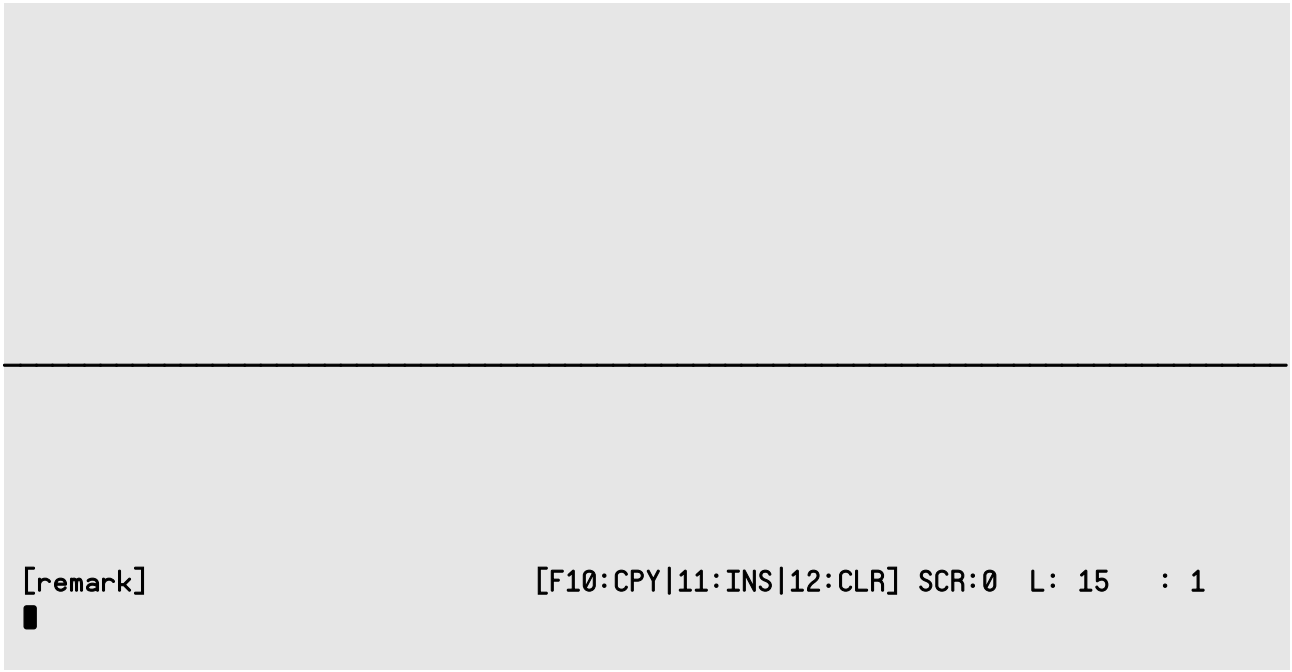
```
screen nr : 0
[sys] ;
cls willkommen zu mental .str cr cr
reboot sys use cls 1 cold w! empty abort ;
bye 0 b! 99 dup b! a! reset ;









-->
```

```
[remark]                        [F10:CPY|11:INS|12:CLR] SCR:0  L: 15   : 1
```

Richtig erkannt: hier haben wir neben anderem Code die Willkommensmeldung des Systems! Wenn **mental** startet, so wird Screen 0 des Tapes **sys** automatisch geladen. Es ist in **mental** problemlos möglich, jeden Screen wie einen Script auszuführen, das System hat also automatisch einen Batchprozessor verfügbar, welcher die gesamte Funktionsvielfalt der Sprache nutzen und diese auch erweitern kann. Was wir konkret sehen, ist die Willkommensmeldung, also ein einfacher String, welcher mit **.str** auf der Konsole ausgegeben wird. In den folgenden Zeilen werden noch die beiden nötige Worte **reboot** und **bye** compiliert, also dem Wörterbuch hinzugefügt. Das geht blizschnell und ist beim Start kaum merklich.

Am unteren Rand des Screens findet sich noch das Wort **-->** welches den Script im nächsten Screen fortsetzt. Schauen wir auch dort mit **ESC** und **ed+** nach:

```
screen nr : 1
usr use 0 load
```



Was geschieht hier? Die Sequenz `usr use 0 load` wählt ein andes Tape zur Bearbeitung aus, und beginnt auch dort ab dem Screen 0 einen Batchdurchlauf.

Nun, damit sind wir bei dem zweiten wichtigen Tape angekommen. Während das sys-Tape mehr statische Systemerweiterungen automatisch hinzufügt und den Startprozess in übergeordneter Weise steuert, ist das usr-Tape als Benutzerbereich gedacht. Hier kann man das aktuelle Projekt bearbeiten. Praktisch: Worte im Screen 0 werden bei Bedarf automatisch mit in das System übernommen. So kann man dort den experimentellen Code speichern, und durch Aufruf des Wortes `reboot` wird durch Ausführung der gesamten Startprozedur das Wort automatisch eingebunden.

Am besten ausprobieren: Nach dem Systemstart ist (deshalb) automatisch das usr-Tape ausgewählt und kann mit `ed0` bearbeitet werden. Einfach ein neues Wort in diesem Screen definieren und mit `reboot` steht das Wort für Experimente zur Verfügung. Achtung: `edwr` nach der Änderung nicht vergessen, sonst wird die Änderung des Screens im Editors ignoriert! ;)

Komfortabler geht das natürlich mit dem Screenbrowser: Nach dem Systemstart mit br den Browser aufrufen, mit Taste `e` den ersten Screen in den Editor laden, bearbeiten und mit `esc` verlassen. Nun kann mit der Taste `w` der Screen gespeichert werden und `r` startet das System neu (Reboot) und die neuen Anweisungen stehen für einen Test zur Verfügung. So kann man sehr schnell neuen Worte definieren und testen. Ist die Programmierung abgeschlossen, können die Screens an einen geeigneten Platz kopiert und archiviert werden.

Mit diesem Startmechanismus steht unter mental ein mächtiges Werkzeug zur Verfügung. So kann bei einem fertigen Programm im sys-Tape automatisch die Verarbeitung gestartet werden, ohne dass `mental` für den Benutzer sichtbar wird - das Resultat verhält sich also wie jedes andere Programm unter TriOS Die bestehenden beiden Tapes stellen dabei nur ein Grundgerüst dar. So kann sich jeder sein ganz individuellen Startvorgang zusammenbauen, kann Anzeigen wieviel Speicher noch verfügbar ist, Zitate und Tageslosungen ausgeben, die Lotozahlen auswürfeln, Sounds abspielen, Mails abfragen (naja, soweit sind wir noch nicht ganz... ;) - der Phantasie ist dort kaum eine Grenze gesetzt. Ist ja alles schließlich noch eine Alphaversion...

MFS, Tapes und Sicherungen

Einige Worte zum Dateisystem: Es ist zwar in einfacher Weise möglich Dateien zwischen **mental**, TriOS oder einem Host-PC zu tauschen, aber dieser Austausch muss ganz bestimmten Randbedingungen gehorchen:

Das wichtigste zuerst: **mental** greift über ein einfaches und sehr schnelles System direkt blockweise auf die Daten in den Tapes zu. Dazu holt es nur den Startsektor des Tapes im FAT16/32-Dateisystem und geht davon aus, dass alle folgenden Sektoren fortlaufend auf dem Datenträger gespeichert sind. Das ist aber nur der Fall, wenn die Daten in den Tapes nicht fragmentiert sind. In einem FAT-Dateisystem ist das aber nur unter bestimmten Bedingungen gegeben: die Tapes müssen als erste Dateien auf eine leere SD-Card kopiert werden. Nur in diesem speziellen Fall wird der Inhalt einer Tape-Datei unter FAT unfragmentiert und mit fortlaufenden Blöcken auf dem Datenträger gespeichert. Wird eine Datei im späteren Verlauf kopiert, verschoben oder verändert, so ist mit großer Sicherheit davon auszugehen, das diese fortlaufende Ordnungsstruktur der Blöcke zerstört wird - die Datei wird fragmentiert.

Dennoch ist dieses System als Kompromiss eine gute Alternative, um in beiden Systemen einfach und schnell auf die Daten zuzugreifen und um mit minimalen Ressourcen auf dem Hive auszukommen.

Sichern und übertragen von Tapes: Von einer SD-Card können unter TriOS mit dem Filemanger oder auf einem Host-PC die Tapes wie normale Dateien kopiert und gesichert werden. Werden sie nur gelesen, was bei einer Sicherung ja der Fall ist, so verhalten sich Tapes wie normale Dateien. Zudem besteht bei einem Lesevorgang nie die Gefahr, dass die Tapes fragmentieren. Die Idee ist letztlich, je ein Tape für ein Projekt zu verwenden. Ein fertiges Programm kann so als einzelne Datei einfach weitergegeben und problemlos eingebunden werden.

Hinzufügen von neuen Tapes: Um einer SD-Card neue Tapes hinzuzufügen, ist es am besten, man stellt die Karte neu zusammen, um eine Fragmentierung zu vermeiden. Im ersten Schritt kopiert man dabei den gesamten Inhalt auf dem Host in ein Verzeichnis auf der Festplatte. Darauf folgend löscht (oder besser formatiert) man die Karte und kopiert die alten und neuen Tapes auf die Karte. In einem letzten Schritt werden die Systemdateien von TriOS oder andere Daten auf das Medium erstellt. Nun kann die Karte mit den hinzugefügten Tapes wieder verwendet werden.

Wer dieses hin- und herkopieren nicht mag, kann sich gleich ausreichend Tapes ablegen und wenn nötig, diese nur passend umbenennen. Ich denke selbst für komplexere Projekte sollten Tapes mit 64 Screens in vielen Fällen ausreichend sein, um den Quelltext ausreichend zu strukturieren und zu erweitern.

Verwendung des Filemanagers unter TriOS: Ohne Host-PC können SD-Karten auch unter TriOS mit dem Filemanager zusammengestellt werden. Dateien können dabei in der RAM-Disk gepuffert werden, deren Inhalt ja auch erhalten bleibt, wenn der Filemanager verlassen wird. So kann ein neues Medium in der Kommandozeile formatiert werden und die benötigten Dateien und Tapes aus der RAM Disk auf der Karte gespeichert werden.

Neue Tapes erzeugen: Mit dem Tool tapecut können unter TriOS neue leere Tapes beliebiger

Größe erzeugt werden. Allerdings müssen die Dateien, sofern sie auf einer fragmentierten Karte erzeugt werden, umkopiert werden, wie es unter dem Punkt "Hinzufügen von neuen Tapes" beschrieben ist. Tapes befinden sich immer nur im Hauptverzeichnis und haben das Schreibschutz-Attribut gesetzt! Das Wort `tapes` unter `mental` zeigt nur Dateien mit gesetztem Schreibschutz an, wodurch mit diesem Attribut alle anderen uninteressanten Dateien ausblenden werden.

Verzeichnisse: mental kann nur auf Dateien im Hauptverzeichnis zugreifen, Unterverzeichnisse werden weder angezeigt noch benutzt.

Hmm, das klingt vielleicht etwas kompliziert, ist aber doch recht einfach in der Handhabung und bietet bei der Verarbeitung enorme Vorteile, die sich noch anderen Stellen zeigen werden. Im einfachsten Fall stellt man sich eine getrennte Karte für `mental` zusammen. Mit einer ausreichenden Anzahl Tapes verschiedener Größe stehen die Chancen sehr gut, dass man nie mehr unter FAT schreibend auf diese Daten zugreifen muss. Das Sichern der Daten stellt dabei kein Problem dar, da Lesevorgänge die bestehende Struktur nicht verändern.