Vierte Dimension

3/2009

Das Forth–Magazin

für Wissenschaft und Technik, für kommerzielle EDV,
für MSR-Technik, für den interessierten Hobbyisten

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

Feldstrasse 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
www.tematik.de

Gegründet 1985 als Partnerinstitut der FH–Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmeßtechnik und bauen z. Z. eine eigene Produktpalette auf.

Know–How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential–Dosierwaagen, DMS–Messverstärker, 68000 und 68HC11 Prozessoren, Sigma–Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,– € im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher“!

RetroForth
Linux · Windows · Native
Generic · L4Ka::Pistachio · Dex4u
Public Domain
`http://www.retroforth.org`
`http://retro.tunes.org`

Diese Anzeige wird gesponsort von:
EDV–Beratung Schmiedl, Am Bräuweiher 4, 93499 Zandt

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690–380
Fax: 02461/690–387 oder –100
Karl–Heinz–Beckurts–Str. 13
52428 Jülich

Automatisierungstechnik: Fortgeschrittene Steuerungen für die Verfahrenstechnik, Schaltanlagenbau, Projektierung, Sensorik, Maschinenüberwachungen. Echtzeitrechnersysteme: für Werkzeug– und Sondermaschinen, Fuzzy Logic.

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10	D–18057 Rostock
Tel.: +49 381 496800-0	Fax: +49 381 496800-29

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: 07044/908789
Buchenweg 11
D–71299 Wimsheim

FORTH–Software (volksFORTH, KKFORTH und viele PDVersionen). FORTH–Hardware (z.B. Super8) und Literaturservice. Professionelle Entwicklung für Steuerungs– und Meßtechnik.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
E–Mail: `Secretary@forth-ev.de`
`Direktorium@forth-ev.de`
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann
E–Mail: `4d@forth-ev.de`

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

Januar, April, Juli, Oktober jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

willkommen zur dritten Ausgabe unseres Forth–Magazins in diesem Jahr.

Anfang September fand die diesjährige EuroForth–Konferenz an der Universität von Exeter im Westen von England statt. Teilnehmer aus den USA, Südafrika, Österreich, Deutschland und aus England waren angereist, um sich über aktuelle Forth–Entwicklungen auszutauschen. Eine Liste der Vorträge und die Online–Proceedings werden auf der Konferenz–Web–Seite `http://www.complang.tuwien.ac.at/anton/euroforth/ef09/` veröffentlicht werden. Zwar nicht mit einem Vortrag selbst vertreten, aber immer wieder diskutiert und als sinnvoll erachtet, ist das von Manfred Mahlow entworfene `PRELUDE`–Konzept zur objekt–orientierten Forth–Programmierung (vgl. auch Heft 2 und 3/2008 unserer VD). Manfred hat mittlerweile eine aktualisierte Version seines `cspForth` unter `www.forth-ev.de` bereitgestellt. Kommende gforth–Versionen sollen ebenfalls das `PRELUDE`–Konzept unterstützen.

Unmittelbar vor der EuroForth–Konferenz fand das Forth2000x–Standardisierungs–Treffen statt. Das Komitee arbeitet an der Verabschiedung des Forth-2009-Standards. Alle wesentlichen inhaltlichen Aspekte sind geklärt (siehe `http://www.forth200x.org/forth200x.html` und die Berichte von Anton Ertl auf Seite 22), das Standard–Dokument geht in die öffentliche Begutachtung und die aktuellen Aktivitäten drehen sich darum, wie der erarbeitete Stand als nationaler (ANSI, DIN, usw.) und internationaler Standard (ISO) anerkannt werden kann. Diskussionen darüber werden im UseNet in `comp.lang.forth` bereits geführt. Im Vorfeld der Forth–Tagung 2010 in Rostock kommt das Standardisierungs–Komitee wieder zusammen, um die Rückmeldungen und Kommentare der öffentlichen Begutachtung zu besprechen. Alle Interessierten sind herzlich eingeladen, teilzunehmen.

Apropos Forth-Tagung: Die Forth-Tagung 2010 findet vom 26. bis zum 28. März 2010 in Warnemünde (Rostock) statt. Der 25. März kann als Zusatztag optional gebucht werden. Aus eigener Erfahrung kann ich das nur empfehlen. Warnemünde, direkt an der Ostsee gelegen, ist einen Extratag auf alle Fälle wert.

Was gibt es im aktuellen Heft zu lesen? Marcel Hendrix berichtet über Sockets und Pipes und wie man sie aus iForth heraus ansprechen kann. Über die Neuerungen der aktuell stabilen Version 0.7.0 von Gforth erfahren wir von Anton Ertl. Willi Stricker beschreibt seinen minimalen Forth–Befehlssatz, den er auch in Hardware realisiert hat. Adolf Krüger und Michael Kalus greifen in ihre Werkzeugkiste und präsentieren uns ihr Dump mit Lineal, um Speicherinhalte bequem anzeigen zu lassen. Über Hard–Disk–Parameter lernen wir etwas in dem 5. Teil von Fred Behringers Artikelserie über Bootmanager und FAT–Reparaturen. Außerdem gibt es noch den diesjährigen Kassenbericht und das Protokoll der Mitgliederversammlung.

Die Abende werden länger und bei Sturm kann man sich gut alten und neuen Projekten widmen. Ich wünsche uns allen einen schönen und ergebnisreichen Herbst.

Ulrich Hoffmann

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://www.forth-ev.de/filemgmt/index.php`

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

Forth auf dem Mac?

PowerMops — das *carbonisierte Mops* für den Mac

Mops war ein objektorientiertes Programmiersystem das aus Neon hervorgegangen ist, einer Sprache von Charles Duff, Vertrieb Kriya, Inc. Kriya hat den Vertrieb von Neon eingestellt, dessen Quellcode der Öffentlichkeit übergeben und lediglich die Namensrechte behalten. So war dann Mops eine komplette Reimplementation des Neon mit etlichen Erweiterungen und selbst dann ebenfalls *public domain*. Es ist dann angepasst worden auf PowerPC unter Mac OS X.

Wer hoffte das es Mops auch noch für die Intel Macs geben würde wird enttäuscht sein. PowerMops generiert PowerPC Maschinencode. In der Intel–Umgebung läuft es unter *Rosetta*[1]. Zwar können Adaptionen für andere RISC Architekturen aus Mops sehr geradeaus und relativ einfach erzeugt werden, aber die altertümliche Intel–Architektur aus den 1980 Jahren ist CISK, und nicht RISC und es wäre abenteuerlich den Code dahin zurück portieren zu wollen, die native Umsetzung für die Intel–Architektur wäre sehr zeitaufwändig. Mike Hore jedenfalls verspürte bisher dazu keinerlei Lust. Falls es aber jemand versuchen möchte, ist es gestattet und würde allseits begrüßt.

PowerMops hat ein ausführliches Manual von Gnarlodious, das er im September 2000 noch in Form von MS–Word Dateien verfasst hatte (Version 4.0), aber dann 2003 in das Webformat brachte, indem er es in TEX gefasst hat[2]. Der *update entry* ist vom 13. Juli 2003 als letztem Stand.

Das Manual enthält vier Teile:

PART I	Intro and Tutorial	21 Lektionen
PART II	Reference	14 Kapitel
PART III	Predefined Classes	12 Kapitel
PART IV	Assemblers	5 Kapitel

Es ist sehr übersichtlich aufgebaut, man findet sich leicht zurecht darin und es bietet selbst Anfängern einen guten Einstieg in die Programmierung der Mac OS X Oberfläche. Alle Operationen die man möchte um in Fenstern zu arbeiten, werden per API vom *Carbon* bereitgestellt. Carbon nennt Apple sein Fenstersystem für die grafische Oberfläche für das OS X. Dieses *Carbonized Mops* macht diese API calls leicht, etliche Beispiele illustrieren das.

Der Name Mops könnte gut ein Akronym sein für *Mikes Objekt orientiertes programmier System* aber Mike Hore fand das die Computerwelt schon genug solche Abkürzungen hat. So liegt der Ursprung der Namensgebung doch im Dunkeln.

Mikes Hoffnung war es das die Mops Benutzer nach und nach ihren Teil beitragen zur Entwicklung von Mops. Es blieb aber doch fast eine Einmann Entwicklung, die es daher schwer hat gegen kommerziell ausgestattete gigantische Entwicklungsumgebungen für den Mac anzukommen. Mike konzentrierte sich auch lieber auf die Entwicklung des Mops Kerns und den basalen Systemcode.

Mops implemented by: Michael Hore
Fähige Assistenz kam von: Doug Hoffman, Greg Haverkamp, Xan Gregg, Nao Sacrada
Dokumentation: Mike Hore, Ed Williams
HTML: Gnarlodious
USENET: comp.lang.forth.mac
Kontakt:

```
email: mike_hore@aapt.net.au
Mops web page: http://www.powermops.org
SourceForge project page:
       http://sourceforge.net/projects/powermops/

Mike Hore lebt in:
Darwin, Northern Territory, Australia.

Right about here ->   _*-_|\
                     /      \
                     \_.--._/
                           v
```

Quelle:

`http://PowerMops.com/MopsManual/MopsManual.html`

Zum Artikel von Willi Stricker im vorliegenden Heft 3/2009

Interessant, die Argumentation von Willi, die mir eine mögliche Antwort auf eine Leserbrief–Frage von mir aus früheren Zeiten gibt: Wie sieht die kleinste Forth–Basis (ich meine *set of primitives*) aus, die es gestattet, ein vollständiges Forth–System (in *high–level*) hochzuziehen? Bei Willi sind es 26 Primitives. Ray Allwright hat in seinem Artikel *From the Net - Minimal Word Sets*, Forthwrite (FIGUK) 95, März 1998, S. 28–32, auf seinerzeit schon bekannte Untersuchungen zu diesem Thema aufmerksam gemacht. Er erwähnt Dan Melchione (eForth verwendet 30 Primitives, es soll aber auch schon mit 11–13 gehen), Peter Lawrence (es geht schon allein mit subtract-and-branch-if-negative). Insbesondere geht Ray Allwright auf die Vorschläge von Bernd Paysan ein: Bernd kommt mit 16 Forth–Worten für eine 2–Stack–Maschine mit 4-Bit-Befehlen aus. In einem weiteren Beitrag, so Ray, verwendet Bernd `NOOP` und `>A` als Präfixe

[1] Neue Programme, die mit dem Symbol *Universal* gekennzeichnet sind, können nativ sowohl auf Intel als auch PowerPC basierten Mac Computern ausgeführt werden. Aber wie sieht es mit Programmen aus, die bereits vorhanden sind? Rosetta ist in Mac OS X integriert, um sicherzustellen, dass die meisten der vorhandenen Programme auch weiterhin verwendet werden können. Rosetta übersetzt im Hintergrund dynamisch die meisten der PowerPC basierten Programme, damit diese auf dem Intel basierten Mac funktionieren. Es findet keine Emulation statt, und es gibt keine Systemumgebung *2. Klasse*. Das Programm wird exakt wie zuvor angezeigt und arbeitet auch wie zuvor. `http://www.apple.com/de/rosetta/`

[2] Gnarlodious schrieb das Manual mit Hilfe des TEX–Edit, das er für einen robusten Script–Editor hält, und meint: „Jeder der HTML auf eine sehr leichte Weise lernen möchte, kann sich gerne die Quellen von der Mops Dokumentation herunter laden. Denn sie sind überaus farbig unterlegt, was leicht macht HTML zu verstehen. Ich finde es ist die einfachste Sache der Welt auf diese Weise HTML und CSS zu erlernen, aber man muss dazu TEX–Edit verwenden.“ Mehr Tips über die Verwendung des iCab/TEX–Edit Teams findet man auf seiner Homepage: `http://Gnarlodious.com/Hello`

für ein 16–Wort–Minimalsystem mit 3–Bit–Befehlen auf einer 1–Stack–Maschine.

Fred Behringer

Forth PARANOIA

Ok, (der Betreff) war ein kleiner Trick, um eure Aufmerksamkeit zu erhaschen. Ich habe das Programm `paranoia` von William Kahan, ein Gleitkomma–Arithmetik–Tester, nach Forth übersetzt. Jemand meinte neulich in dieser Diskussion hier, dass wir nicht mehr den Stand von 1985 haben, was Gleitkomma–Berechnungen angeht. Aber die Sachlage ist doch undurchsichtig, wenn es um Forth und Gleitkomma geht.

Ich habe `paranoia` auf einigen Forth–Systemen laufen lassen.

Die guten Nachrichten:

> Gforth und PFE haben alle Tests bestanden und deren Gleitkomma–Arithmetik erhielt vom Programm die Auszeichnung *Excellent* und die Art zu Runden sei konform zum IEEE Standard P754. kForth auf ppc/OSX erzielte ähnliche Ergebnisse in einem Test von David Williams.

Die schlechten Nachrichten:

> Beim kForth auf x86 wurden je ein *Defekt* und ein *Makel* aufgedeckt, und das Programm hielt dessen FP–Arithmetik für *akzeptabel*. Und noch einige gut bekannte Forth–Systeme waren in dem Test unterschiedlich erfolgreich.

Wobei man berücksichtigen muss, dass auch noch Fehler in der Forth–Übersetzung lauern können. Denn es war schwierig, dieses Programm zu portieren, wegen der Länge, und weil besonders sorgfältig vorgegangen werden musste, um nicht unabsichtlich arithmetische Ausdrücke zu entstellen.

Hier der Link zum Programm:
`ftp://ccreweb.org/software/fsl/extras/paranoia.fs`

Es sollte auf Standard–94–Forth–Systemen laufen, mit oder ohne seperatem FP–Stack. Bitte teilt mir mit, falls ihr irgendwelche Fehler findet.

Krishna Myneni, comp.lang.forth, 20.05.2009

(Übersetzung: mk)

Interview mit Dirk Brühl

Michael (mk): Dirk, du hast etliches in Forth programmiert und beruflich längere Zeit davon gelebt. Wen kennst du eigentlich, der Forth ebenso verwendet hat?

Dirk (db): Mmmmh ... muss ich drüber nachdenken! Ich kenne etliche, die von Forth begeistert sind, aber wenige die Forth tatsächlich auch verwendet haben. Persönlicher bekannt war mir Wolfgang Allinger, der in Forth programmierte, Software für Ultraschallgeräte bei Krautkrämer (`http://www.ob-ultrasound.net/history1.html`) und Dr. Delius. Er war von Forth so begeistert, dass er später alle seine DNA–Analysen mit Hilfe von Forth durchgeführt hat. Und Dr. Haeggqwist, der war da eine Ausnahme, lebte nicht direkt davon.

mk: Kanntest du nicht auch Heinz Schnitter?

db: Ja, natürlich. Er verwendete im Gegensatz zu mir und den Vorgenannten Forth im großen Stil, zum Beispiel etwa hundert Mikroprozessoren, in Forth programmiert, miteinander verbunden, um einen Beschleuniger in Fahrt zu bringen. Ich habe ihn stets bewundert. Er hat mich damals animiert, für ein Projekt vier Mikroprozessorboards miteinander zu koppeln, und auf der Forth–Tagung in München habe ich einigen Interessierten gezeigt, wie auf einfache Art und Weise, mit einem Flachkabel, zehn Mikroprozessorboards mit RSC–Forth dazu gebracht werden können, miteinander zu kommunizieren.

mk: Forth ist ja kein Gedanke in nur einem einzelnen Hirn geblieben. Und auch die Umgebung des Forth hat sich doch stark geändert, *normative Kraft des Faktischen* sozusagen, so dass das Forth, wie es mal war, da verloren drin aussieht, oder?

db: Das ist nur der äußere Schein, vor allem in der PC–Welt. In der Welt der Mikroprozessoren ist Forth gut verbreitet — vielleicht nicht so sehr in Deutschland oder in den USA — das Land, wo Forth ohne Vorurteile verwendet wird, ist wohl am ehesten England — siehe MPE und Inventio Software. Auch in Russland ist Forth stark verbreitet, so wie es in der ehemaligen sogannten DDR war. Forth arbeitet hauptsächlich im Verborgenen, wie Du Julian V. Nobles Tutorial entnehmen kannst.

mk: Wie du ja selbst schon gesagt hast, für die einfachen Mikroprozessoren damals war es passend und Klasse.

db: Für die einfachen Mikroprozessoren wie dem 6502 hätte ich Forth nicht gebraucht. Da konnte ich mir selbst helfen und habe ich auch gemacht. Sogar für die Geldkarten–Softwareerweiterung für SIEMENS–Parkscheinautomaten in den 90er Jahren habe ich wieder alles in Assembler geschrieben, weil die ursprüngliche Software in Assembler geschrieben war, aber sehr gut strukturiert. Nur deshalb war es mir möglich, Erweiterungen problemlos einzufügen. Auch die Software für die Burr–Brown–Terminals TM71 etc. habe ich komplett in Assembler geschrieben, ursprünglich war die Software in C, aber die wollte Burr-Brown USA für die Magnetkarten–, Barcodeleser– und Drucker–Erweiterungen (hier in der BRD) nicht rausrücken — zu meiner Freude, denn mit C wäre ich aufgeschmissen gewesen. Auch haben sie behauptet, das Projekt wäre nicht durchführbar, man würde einen zweiten Mikroprozessor dafür brauchen, was sie dann auch gemacht haben. Aber in Deutschland wurden die Terminals mit meiner Software verkauft, die nur einen, den vorhandenen, Mikroprozessor benötigte. Das war in den 80er Jahren,

ebenso die Software für Feuerwehr, U–Bahn– und Straßenbahn-Funkverkehr war in Assembler geschrieben, allerdings unter Verwendung meines eigenen virtuellen Mikroprozessors, wie ich bereits mal erwähnt hatte.

Für die komplizierten Mikroprozessoren ist Forth wesentlich wichtiger, für mich sogar unabdingbar! Nur mit Forth konnte ich ein Projekt für Philips mit einem 68331 und einem Xilinx-FPGA durchführen. Deshalb hatte ich für das Layout des Boards Leute beauftragt, die mit UR/Forth arbeiteten und für mich dann ein Forth-System auf den 68331 portiert haben, damit ich die Leiterplatte testen konnte. Ich musste dann letztendlich das Leiterplatten-Layout selbst machen, weil beim ersten Mal zu viele Fehler drin waren, aber dank Forth konnte ich dann die Platine komplett testen. War mein *Prunkstück*, ein Ausschnitt davon mit meinem Layout auf SophistiCAD (ein in Forth geschriebenes Leiterplatten-Layout-System) ist auf `http://www.bruehlconsult.com/de.html` zu sehen.

Mein letztes kommerzielles Projekt war eine Dual–Prozessor–Architektur mit einem 8–Bit 6502–kompatiblen Mitsubishi–Mikroprozessor (auf den ich das RSC–Forth portiert habe; siehe `http://www.bruehlconsult.com/Downloads-de.html`) und einem 32–Bit–Philips–ARM–LPC–Mikroprozessor, das ich nur deshalb durchführen konnte, weil ich von MPE ein Board mit diesem Mikro und integriertem Forth bestellen konnte.

Der Mitsubishi M16, den ich zunächst für das Projekt vorgesehen hatte — ich hatte sogar etliche Entwicklungsboards mit dem M16 — ist da durchgefallen, weil es kein Forth dafür gab und weil ich doch nicht den Nerv hatte, ein eigenes Forth dafür zu schreiben. Einen Werkstudenten aus Stuttgart konnte ich beauftragen, die PWM–Funktion des M16 auszuprobieren (er konnte C), aber das hat dann auch nicht mehr geholfen, weil ich feststellte, dass eine 16–Bit– Timer–Auflösung für unsere Ansprüche zu gering war. Also war ich froh, dass es Stephen Pelc und sein MPE–ARM–Forth gab.

Ansonsten hätte ich — zum ersten Mal in meinem Leben und ausgerechnet hier in der Fremde, das Handtuch werfen müssen. Mit Hilfe von Forth lief alles bestens — das war das Projekt, in das ich zum Testen eines PID–Reglerprogramms ein 8–Kanal–Oszilloskop, nur aus Software bestehend, eingebaut hatte.

Meine Quintessenz: Je komplizierter das System, umso mehr habe ich davon, wenn ich Forth einsetzen kann — auch wenn das nicht die gängige Meinung ist, es ist meine persönliche Erfahrung. Und zu den komplizierten Systemen zählt natürlich auch der PC, auch wenn ich in der Grundig–Akademie bei meinem letzten Mikroprozessorkurs Mikroprozessor–Programmierung mit Hilfe des PC–Debuggers unterrichtet hatte. Ich würde aber niemals auf die Idee kommen, ein PC–Programm in Assembler oder C zu schreiben. Eine Ausnahme war VB — und das führte zum Wunsch nach einem visualForth!

mk: Ich bin ja immer wieder verwundert, dass Forth trotz der Hardwareentwicklungen in all den Jahren weiter besteht — wenn auch fast nicht wiederzuerkennen unter all den Objekten in modernen großen Forth–Systemen. Das win32forth kann ich nicht mehr komplett verstehen so wie das RSC–Forth einmal.

db: Wie aus meinen obigen Ausführungen zu entnehmen ist, Forth existiert weiterhin nicht trotz der Hardwareentwicklungen, sondern gerade wegen der Hardware–Entwicklungen — siehe die Entstehungsgeschichte des OpenBoot–PROMs. Ich hätte gerne mit dem BigForth gearbeitet, aber Bernd Paysans Sachen waren für mich zu undurchschaubar — aber der Drache ist prima! Hätte gerne auch mal so etwas gemacht, aber wie? Dazu muss man wahrscheinlich ein Bernd Paysan sein!

Ich gebe mir auch keine Mühe, Win32Forth [1] komplett zu verstehen. Meine Arbeitsgrundlage ist, dass w32f ein Forth–System ist, und mir die Forth–System–Worte zur Verfügung stellt — das ist gerade *the beauty of Forth*. Forth ist ein Knowledge–System, ein System, dass sich selbst kennt, und dessen eingebautes Wissen mir hilft, meine Aufgaben zu erfüllen. Dazu kommt, dass ich stets nur an einem kleinen Stück, einem Modul, einem Teil der Software arbeite. Das entlastet meinen Denkapparat, ich kann mich auf dieses kleine Stück konzentrieren, und von dem Rest muss ich nur wissen, dass er da ist und funktioniert, und das ist durch die inkrementelle Arbeitsweise sichergestellt.

Damit kann ich arbeiten, solange ich mich in bekanntem Forth–Terrain bewege. Und wenn es darum geht, neues Terrain zu betreten, gibt es Hilfen weltweit. Im Falle der Grafik konnte ich damals auf Jos van de Vens Ressourcen zurückgreifen, und das war phantastisch! Ich brauche nicht zu wissen, wie das Forth gefädelt ist. Die CPUs sind heutzutage so schnell, dass es für mich sowieso keine Rolle spielt. Virtuell ist virtuell, und virtuell ist nun Mainstream geworden — das nur als Randbemerkung.

mk: Du hast Forth also als Instrument benutzt, um Einsicht zu bekommen in Maschinen, die dir bis dahin unbekannt waren. Als Türöffner sozusagen?

db: Im Falle des Projekts mit dem 68331 auf jeden Fall. Und mit dem ARM–Mikroprozessor hätte ich ohne Forth überhaupt nicht arbeiten können, obwohl ich dort ein kleines Modul in Maschinensprache geschrieben habe, nur drei oder vier Maschinenbefehle lang. Auch jemand anders, der da intelligenter ist als ich, bräuchte trotzdem wesentlich mehr Zeit, weil er erst einmal jede Menge über die internen Register und die Start–Routinen lernen muss, was mit Forth nicht der Fall ist. Für den PWM–Timer habe ich ein Demoprogramm von Philips, das in C geschrieben war, in Forth transferiert, sah danach viel besser aus und

war einfacher zu handhaben in Forth. Was das RSC–Forth angeht: So einfach sollte es mit visualForth wieder werden!

mk: Ich finde, es braucht ein didaktisches Durcharbeiten der Schichten des Forth, damit junge Leute damit wieder einsteigen können in das, was man Programmieren nennt.

db: Ich lese derzeit ein interessantes Buch *MADE to STICK — Why some Ideas Survive and Others Die*, von Chip Heath & Dan Heath, außerordentlich interessant. Sie schreiben zum Beispiel vom *Fluch des Wissens* — wenn jemand etwas weiß, kann er es meist einem Nichtwissenden nicht vermitteln, weil er inzwischen zu abstrakt denkt (in Kurzform). Das erinnerte mich an meine These, dass es immer nur eine gewisse Anzahl von Menschen gibt, die sich auskennen, und wo bleiben die anderen? Für die anderen will ich sorgen. Meine Devise war immer, dass Forth für den Mann an der Werkbank ist.

mk: Motivation hast du ja genug. An etlichen Stellen ist dir der Ruf nach so etwas wie visualForth in den Ohren geklungen. Das hilft, um an so einem Projekt nicht so schnell zu verzweifeln, denke ich. Denn das ist ja kein so ganz kleines Vorhaben.

db: Ja, das muss ich mir immer wieder erzählen, obwohl es mein eigener dringender Wunsch war, endlich da mal was mit Windows machen zu können, ohne das nervtötende VB verwenden zu müssen. Und dass andere auch danach suchten, hätte mir klar sein müssen, und das kam ja dann auch nach einiger Zeit raus. Die Forth–Tagung (2009 in Neuenkirchen bei Rheine) und die Unterstützung der FG haben mir außerordentlich geholfen. Durch diesen Ecktermin kam alles auf die Reihe, pünktlich und ohne Stress [2].

mk: Dirk, hab Dank für die Mitteilung deiner Erfahrungen.

Dirk Brühl lebt heute im Ruhestand in Pensylvania USA.

Links

[1] `http://tech.groups.yahoo.com/group/win32forth/`
[2] `http://www.visualforth.org/Alpha`

Josephspfennig in Heft 1/2009

Hallo VD–Redaktion, ich möchte noch einmal auf den Artikel über den Josephspfennig in Heft 1 dieses Jahres zurückkommen. Dort heisst es:

> Aus 1 Pfennig im Jahre 0 zu 5% mit Zins und Zinseszins ergab sich im Jahr 2003 der rechnerische Betrag von: 27.679.996.896.051.261.677.068.884.476.135.650.875.110,12 DM

die Angabe von 12 Pfennig lässt vermuten, dass das Ergebnis auf eine halben Pfennig genau ist, das ist aber mit nichten so!

Löst man das Josephspfennig–Problem geschlossen, so folgt man der Zinseszinsformel:

$$K_n = K_0(1+z)^n$$

wobei K_0 das Anfangskapital (hier 1 Pf), z der Zinssatz (hier $5\% = \frac{5}{100}$), n die Laufzeit (hier 2003 Jahre) und K_n schließlich das Endkapital (unsere gesuchte Größe) ist. Rechnet man für den Josephspfennig in Pfennigen, wird die Formel also zu

$$K_{2003} = 1 \cdot \left(\frac{105}{100}\right)^{2003}$$

Diese Formel gilt es also exakt auszurechnen. Hmmm — Lisp kennt exakte Brüche:

```
(setq Kn (expt 105/100 2003))
```

Das exakte Ergebnis ist ein Bruch, dessen Nenner (selbst nach Kürzen des Bruchs) 2606 Ziffern besitzt und der daher hier besser nicht abgedruckt wird. Wir sind aber ja nur auf den Pfennig genau an dem Ergebnis interessiert, weswegen wir den Bruch auf die nächstliegende ganze Zahl runden können: `(round Kn)`. Das liefert uns

27679996896157634534421221977463776885435584 Pf.

Dieser Wert weicht vom Ergebnis aus Heft 1/2009 um

```
    27.679.996.896.157.634.534.421.221.977.463.776.885.435,84 DM
-   27.679.996.896.051.261.677.068.884.476.135.650.875.110,12 DM
=          106.372.857.352.337.501.328.126.010.325,72 DM
```

ab. Das Ergebnis aus Heft 1/2009 ist also um rund 10^{29} DM zu klein. Her damit! Kein Wunder, dass wir Finanzkrise haben :-)

Wer schreibt eine exakte Brucharithmetik für/in Forth?

Es grüßt Euch, Johannes Brooks

Leitungen und Stecker — Sockets und Pipes in Forth

Marcel Hendrix

Vorwort

Hast du dich schon mal gefragt, wie du andere Programme auf deinem Computer steuern kannst? Oder wie Programme auf verschiedenen Rechner das untereinander bewerkstelligen — also im lokalen Netz oder im Internet? Dann stößt du auf das, was man eine API (application programming interface) nennt. Fragt sich nur, wie man von Forth aus solche APIs anderer Programme erreicht. Freundlicherweise ist diese Kommunikation zwischen den Prozessen inzwischen textbasiert gelöst. Es werden Anweisungen oder Skripte ausgetauscht. Und dazu haben diese Prozesse (Programme) Stellen, an denen man sich bildlich gesprochen *anstöpseln* kann. Eine einführende Erklärung dazu habe ich bei Marcel Hendix zum iForth gefunden. Sein englischer Text ist hier übersetzt ins Deutsche wiedergegeben, und wurde von mir um ein erklärendes Glossar ergänzt.

Ihr werdet sehen, dass seitdem der Beitrag verfasst wurde, einige Jahre vergangen sind. Doch da Grundlagen behandelt werden, ist das Thema weiterhin aktuell. Inzwischen gibt es Windows XP und Windows Vista, und Windows 7 ist angekündigt. Linux ist fortgeschritten, und es gibt Mac OS X mit Intelprozessoren. All diese sind hier nicht extra erwähnt, der Code läuft aber auf diesen Plattformen mit iForth 3.0, final 32–bit release, datiert vom 17. Mai 2008. Es ist erhältlich für Windows, Linux x86_32 oder x86_64, und Mac OS X (Intel hardware only). iForth wird derzeit für 100 € vertrieben und unterstützt. Es bietet drei Fenster: Die klassische Konsole, dann einen RTF–Editor, und den *graphics terminal screen* in OpenGL–Grafik (sehr schön!). Dazu eine Erweiterung um parallel processing, und ein Metacompiler, der schnelle und plattformunabhängige 32-Bit–Forth–Systeme erzeugt. Unter Linux und Windows gibt es Unterstüzung für (named) pipes, dynamic linking, Tcl/Tk, (X-)windows, SVGA graphics und full digital sound. Tcl/Tk arbeitet spektakulär gut zusammen mit iForth unter Windows. Diese I/O–Funktionalität wird durch ein Server–Programm — den iServer — erreicht, das in C geschrieben ist und mitgeliefert wird. Womit wir wieder bei den Steckern wären... M. Kalus

Die Stecker

Hier eine *volkstümliche* nicht ganz exakte Beschreibung (helft mir mit einer bessern aus — email!). Das Konzept der *Stecker* (sockets) ist ein Weg, eine Kommunikation zwischen Prozessen herzustellen (IPC; inter process communication). Es befähigt Programme auf demselben Computer, sich zu finden und miteinander zu reden. Aber das ist nicht alles. Stecker funktionieren auch für Programme, die in einem Netzwerk laufen — überall im Internet, falls du so verdrahtet bist.

Um Stecker für eine IPC zu haben, musst du Dienstprogramme und Nutzerprogramme schreiben (server and client program). Der Dienst (server) läuft auf deinem eigenen (local) Computer und ist unter deiner Kontrolle. Die Nutzerprogramme (clients) hingegen können irgendwo im Netzwerk sein, auch auf deinem eigenen Computer als spezieller Fall, und brauchen nicht von dir selbst gemacht worden zu sein. Der Dienst horcht an einem Eingang (port) nach Clienten die in Kontakt kommen wollen. Solche entlegenen (remote) Clienten müssen den Namen deines Computers und die Nummer des Eingangs wissen, an dem dein Dienst wartet. Populäre Beispiele für Dienste auf fast allen Computern sind `FTP` (port 20), `TELNET` (port 23), `FINGER` (port 79), `TIME-OF-DAY` (port 13) und `QUOTE` (port 17). Dein eigener Dienst nimmt einfach eine Eingangsnummer, die gerade frei ist, zum Beispiel 3145. Die meisten Stecker–IPCs senden Zeichenketten (strings) zwischen Dienst und Nutzer hin und her und folgen dabei irgendeinem der veröffentlichten Vorschriften (protocol). Für den eigenen Gebrauch hingegen kann man sich einfach selbst was ausdenken, das klappt. Und weil man Forth benutzt, ist das Protokoll der einfachste Teil: Zeichenketten senden (counted strings)[1] und `EVALUATE` benutzen geht ganz nett.

Einen Dienst auf deinem Computer zu starten, hat dann die simple Form:

```
3145 myserver
```

Dabei ist 3145 die Eingangsnummer, an der dein Dienst horcht[2] . Und `myserver` ist ein Forth–Wort (colon definition), das vielleicht eine Endlosschleife enthält. So prüft es, ob ein Client am Eingang 3145 Kontakt sucht,

[1] COUNTED STRINGS sind Zeichenketten, in denen das erste Byte die Längenangabe der Kette enthält. Die Empfangsroutine legt solch eine Zeichenkette im Speicher ab und hinterlegt auf dem Stack die zugehörige Speicheradresse und die Länge der Zeichenkette (– adr u). EVALUATE nimmt dann diese Daten vom Stack, um die Zeichenkette auszuwerten. So kann EVALUATE auf alle möglichen Speicherstellen zugreifen, um dort in Zeichenketten abgelegte Anweisungen auszuführen. EVALUATE ist somit vektorisiert über den Datenstack des Forth. Dieser Mechanismus ist im Forth schon implementiert und braucht daher für eigene Server nicht erst erfunden zu werden.

[2] Die Nummer ist für das Beispiel willkürlich gewählt worden.

und tauscht dann Zeichenketten mit dem Clienten, wobei ein selber entworfenes Protokoll benutzt werden kann — Vorschriften dafür gibt es nicht. Die Kommunikation durch diese Stecker geht in beide Richtungen (bidirektional).

Die Aufgabe für den Clienten ist genauso simpel. Nehmen wir an, der Client liefe auch auf dem eigenen Computer, dann könnte man so starten:

```
S" //." 3145 myclient
```

oder mit

```
S" frunobulax" 3145 myclient    ,
```

wenn dein Computer `frunobulax` heißt. Neuerzeits kennt iForth das Wort *HOSTNAME (– addr u)*, welches automatisch der Computernamen herausfindet und auf dem Stack ablegt. Jede URL ist ok, solange du eine gültige Port–Nummer angibst und dein Server dahinter aktiv ist.

Bis du hierher den Ausführungen gefolgt, kannst du mehr darüber lernen, indem du dir die Dateien holst, die oben genannt worden sind. Denn damit wird iForth um die folgenden Worte erweitert:

`OPEN-SERVICE ( c-addr u port# -- socket )`
example: `S" frunobulax" 3145 OPEN-SERVICE ( -- 4 )`

`CREATE-SERVER ( port# -- lsocket )`
example: `3145 CREATE-SERVER ( -- 16 )`

`LISTEN ( lsocket /queue -- )`
example: `16 7 LISTEN`

`ACCEPT-SOCKET ( lsocket -- socket )`
example: `16 ACCEPT-SOCKET ( -- 3 )`

`READ-SOCKET ( socket c-addr maxlen -- c-addr size )`
example: `3 PAD 125 READ-SOCKET ( -- 'pad 125 )`

`WRITE-SOCKET ( c-addr size socket -- )`
example: `PAD 128 3 WRITE-SOCKET`

`CLOSE-SOCKET ( socket -- )`
example: `3 CLOSE-SOCKET`

`+CR ( c-addr1 u1 -- c-addr2 u2 )`
appends a hard CR+LF pair to the text

`BLOCKING-MODE ( socket on/off -- )`
put the socket in/out of blocking mode

`SET-SOCKET-TIMEOUT ( u -- )`
set the global timeout (in ms) on reading sockets

`GET-SOCKET-TIMEOUT ( -- u )`
get the global timeout (in ms) on reading sockets

Die Röhren (Named Pipes)

Noch eine volkstümliche unexakte Beschreibung. (Hast du eine bessere? Email mir!) Eine *benannte Röhre* oder Leitung ist der zweite Weg, Interprozesskommunikation (IPC) zu betreiben. Programme auf dem selben Computer (Linux), oder irgendwo im Netzwerk oder Internet[3], könne sich gegenseitig ausfindig machen und miteinander reden. Der Unterschied gegenüber dem Stecker–Modell ist, dass man keine von diesen ziemlich willkürlich festgelegten Eingangsnummern benutzen muss, dafür aber einen (ebenso willkürlichen :-) Dateinamen. Und ich kenne bisher auch keine öffentlich erreichbaren Röhren die im Internet einen Namen hätten. Wie bei den Steckern so schreibt man auch hier Forth–Programme für beide Seiten, den Dienst und den Clienten, und das IPC arbeitet ebenfalls mit Zeichenketten, die hin und her gehen und die einem Protokoll folgen, das man selbst entwirft.

Einen solchen benamten Röhren–Dienst auf deinem Computer zu starten, hat dann die simple Form:

```
S" //./pipe/forthserver" serverspeed ( NT )

S" /dev/forthserver" serverspeed  ( Linux )
```

Hier definiert der Name den Pfad, den der Client nehmen muss, um zu deinem Server zu gelangen (auch dieses Beispiel ist erfunden). Beachte den Teil, der da `/pipe` heißt, das ist für Windows so erforderlich. Das Beispiel–Forthwort `serverspeed` wiederum enthält auch — möglicherweise — eine Endlosschleife. Sie prüft, ob ein Client den Kontakt sucht, und tauscht dann wieder Zeichenketten mit dem Clienten aus, wieder in einem eigens ersonnenen Protokoll. Auch die Kommunikation in den Pipes ist bidirektional.

Und auch der Teil für den Clienten ist einfach. Falls der Client auch auf deinem Computer läuft, starte ihn z. B. so:

```
S" //./pipe/forthserver" clientspeed
```

oder

```
S" //frunobulax/pipe/forthserver" clientspeed
```

oder

```
S" /dev/forthserver" clientspeed    ,
```

falls dein Computer wieder `frunobulax` heißt. (Linux unterstützt allerdings keine Pipes die über ein Netzwerk hereinkommen.)

Auch nun kannst du mehr dazu erkunden, indem du die Dateien holst, die sich mit den *named pipes* befassen. Und iForth wird dadurch um folgende Worte erweitert:

`OPEN-NAMED-PIPE ( c-addr u timeout -- handle )`
example:
`S" //frunobulax/pipe/clientspeed" 5000 OPEN-NAMED-PIPE`

`CREATE&ACCEPT ( c-addr u -- handle )`
example:
`S" //frunobulax/pipe/serverspeed" CREATE&ACCEPT`

`READ-NAMED-PIPEX ( c-addr maxlen handle -- c-addr size )`
example: `PAD 125 3 READ-NAMED-PIPEX ( -- 'pad 125 )`

`WRITE-NAMED-PIPE ( c-addr size handle -- )`
example: `PAD 128 3 WRITE-NAMED-PIPE`

`CLOSE-NAMED-PIPE ( handle -- )`
example: `3 CLOSE-NAMED-PIPE`

[3] Windows ließ dies auch mal zu, aber nur für seine NT Dateisysteme.

Vielleicht kannst du diesen Satz an Forthworten etwas leichter verstehen als das *socket wordset*, aber die Stecker sind universeller einsetzbar, sie können jede URL benutzen.

Systeme, auf denen iForth sockets und pipes unterstützt

Windows NT, XP, Vista, 7
Beides, das socket und das pipe wordset, funktionieren mit Windows in jeder Version.

Eigentlich bräuchte das iForth gar keine speziellen read/write/close Wörter für Linux, denn die Schnittstellen (handles und sockets) sind kompatibel mit den Worten, die auch für das Dateisystem da sind — `READ-FILE` `WRITE-FILE` und `CLOSE-FILE`. Aber wegen des Microsoftismus funktioniert es so nicht unter Windows. Deshalb hat das iForth andere Namen für die read/write/close Aktionen genommen, damit die Unterschiede dahinter einfach auseinandergehalten werden können.

Das named pipe wordset ist auf einzelne Clienten, die mit einem einzelnen Server verbunden sind, beschränkt. (Eigentlich sollte Windows multiple Clienten unterstüzten, aber es war nicht herauszufinden, wie es das macht.)

Linux 2.0.x und Mac OS X (Intel)
Beides, das socket und das pipe wordset, funktionieren mit Linux jeder Version. Beim Linux sind die pipes nicht so nützlich, da sie dort im Netzwerk nicht funktionieren. Aber man kann hier mehrere Klienten an einen Server anbinden. Die pipes wurden auf sockets und dem etwas modifizierten Code von W. Richard Stevens *Advanced Programming in the UNIX Environment* aufgebaut[4].

Die Leitungsfähigkeit

Es gab einen erkennbaren Unterschied in der Performance zwischen sockets und named pipes. Die Tests wurden folgendermaßen durchgeführt (Stand 09.06.2009):

Benchmark 1: 20 Mbytes kopiert zwischen zwei iForths auf Maschine 1;
Benchmark 2: 20 Mbytes kopiert zwischen einem iForth von Maschine 2 an ein iForth auf Maschine 1.

Systeme für den sock–Test:
Windows XP Pro, einmal (1) auf einem Intel PIV 3GHz/1GB und (2) auf Intel Core 2 Duo 2.66 GHz/2GB. Die PCs waren verbunden über motherboard Realtek network adapters (100 Mbit/s).

Systeme für den npipe–Test:
Windows NT 4.0, einmal (1) auf Intel Pentium 166MHz/48MB und (2) auf Intel Pentium 200MHz/48MB. Die PCs waren im Netzwerk verbunden über preiswerte NE2000 clones (10 Mbit/s).

benchmark	process A read	process B write
sock bm 1	72 MB/sec	732 MB/sec
sock bm 2	11.5 MB/sec	2.857 GB/sec
npipe bm 1	15 MB/sec	15 MB/sec
npipe bm 2	715 KB/sec	715 KB/sec

Die Tests wurden auch unter Linux durchgeführt und verliefen noch etwas besser dort.

Wie man sieht, arbeiten die sockets schnell bei Prozessen auf verschiedenen Maschinen. Hingegen schnitten die pipes bei Prozessen, die auf demselben Rechner lagen, besser ab als wenn sie im Netzwerk verwendet wurden. Unter Linux gibt es fast keinen Unterschied zwischen sockets und pipes.

Ich rate dazu, sockets zu verwenden. Sie sind portabel, schnell, und arbeiten im Netzwerk ebenso wie innerhalb des Betriebssystems[5].

Der Quellcode

In seinem WEB–Beitrag gibt Marcel Hendrix links an, unter denen man, je nach Betriebssystem, die passenden Quellen findet, um sockets und pipes in iForth realisieren zu können. Dazu wird jeweils ein Stück C–Code benötigt (iForth Server C–part) um die basale Anbindung von iForth an das Betriebssystem aufzubauen. Und dann ein Stück Forthcode, um diese C Funktionen mit dem Forth Socket Wordset zu verbinden. Der folgende screenshot zeigt, was geboten wird.

Crystal Clear filesystem socket (Quelle: wikimedia.org)

[4] Advanced Programming in the UNIX Environment. (Addison–Wesley Professional Computing Series) von W. R. Stevens und Stephen A. Rago von Addison–Wesley Longman, Amsterdam (Gebundene Ausgabe — 7. Juli 2005); bei Amazon.de erhältlich.

[5] Unglücklicherweise funktionieren pipes mit Namen nicht unter Windows XP Pro. Die Windows SDK informiert darüber, dass dafür mindestens der Windows 2000 Server benötigt wird. Daher sind die pipes in ihrer Nützlichkeit doch erheblich eingeschränkt, und Marcel Hendrix überlegt derzeit (mail 9.6.2009) ob in zukünftigen Versionen diese im iForth eingebaute Unterstützung nicht ganz entfallen sollte.

Links

Marcel Hendrix
`http://home.iae.nl/users/mhx/pipes%26socks.html` `http://home.iae.nl/users/mhx/i4faq.html`

Glossar

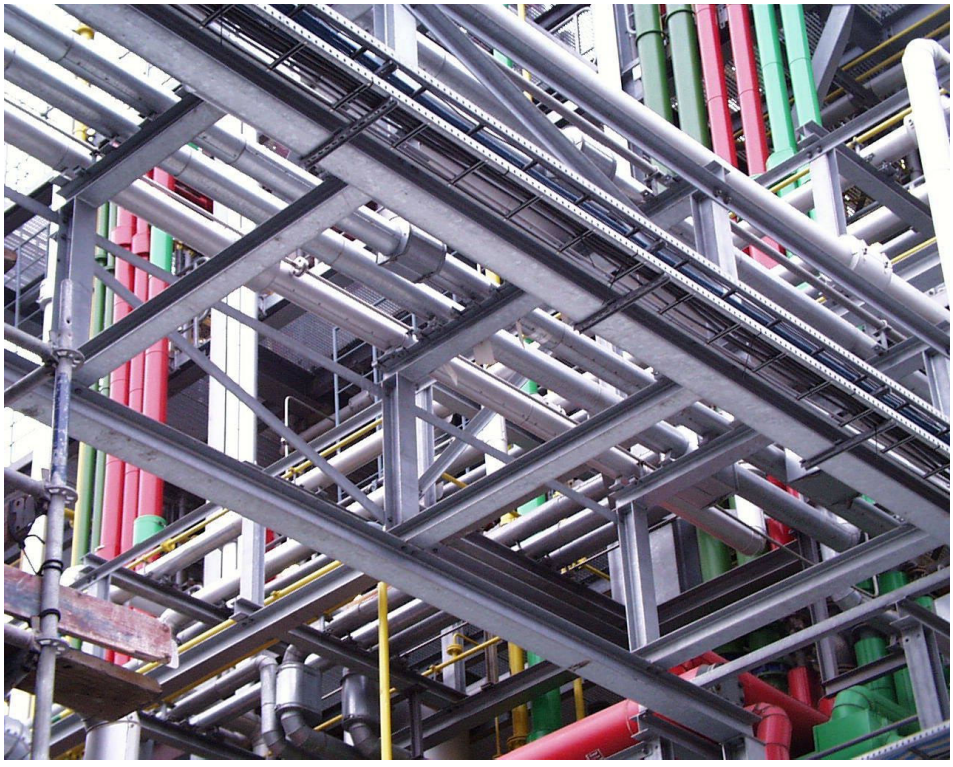
Socket Ein Socket (engl.; Sockel oder Steckverbindung) ist eine bidirektionale Software-Schnittstelle zur Interprozess- (IPC) oder Netzwerk-Kommunikation. Sockets sind vollduplexfähige Alternativen zu Pipes oder Shared Memory. Sockets bilden eine plattformunabhängige, standardisierte Schnittstelle (API) zwischen der Netzwerkprotokoll-Implementierung des Betriebssystems und der eigentlichen Anwendungssoftware.

API Eine Programmierschnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Oft wird dafür die Abkürzung API (für engl. application programming interface, deutsch: „Schnittstelle zur Anwendungsprogrammierung“) verwendet. Im Gegensatz zu einer Binärschnittstelle (ABI) definiert eine API die Verwendung der Schnittstellen auf Quelltextebene. Programmierschnittstellen werden in folgende Klassen einteilen: funktionsorientiert (z. B. Dynamic Link Library) dateiorientiert (z. B. Gerätedateien unter UNIX) objektorientiert (z. B. ActiveX-DLLs) protokollorientiert (z. B. FTP)

Pipe Die Pipe (englisch für Rohr, Röhre) bezeichnet einen gepufferten uni- oder bidirektionalen Datenstrom zwischen zwei Prozessen nach dem „First In – First Out“-Prinzip. Das heißt vereinfacht, dass die Ausgabe eines Prozesses (ein Programm in Ausführung) als Eingabe für einen weiteren verwendet wird.

URL Als Uniform Resource Locator (URL, dt. „einheitlicher Quellenanzeiger“) bezeichnet man eine Unterart von Uniform Resource Identifiern (URIs). URLs identifizieren und lokalisieren eine Ressource über das verwendete Netzwerkprotokoll (beispielsweise HTTP oder FTP) und den Ort (engl. location) der Ressource in Computernetzwerken. Da URLs die erste und häufigste Art von URIs darstellen, werden die Begriffe häufig synonym verwendet. In der Umgangssprache wird URL häufig als Synonym für Internetadresse verwendet.

(Quelle: Wikipedia)



1998-11-16-Pipes (Quelle: wikimedia.org)

Gforth 0.7.0

Anton Ertl

Was ist Gforth?

Gforth ist ein freies, portables, und schnelles Forth-System. Gforth implementiert ANS-Forth und die bis jetzt beschlossenen Erweiterungen von Forth200x.

Freiheit Gforth wird unter der GNU General Public License v3 verteilt, und die Benutzer können Gforth laufen lassen, kopieren, weiterverbreiten, studieren, ändern und verbessern, und auch die geänderten Versionen weiterverbreiten.

Portabilität Gforth läuft auf diversen Unix-Varianten (0.7.0 wurde auf Linux, Solaris, und HP/UX getestet), auf Windows, und MacOS X. Weiters gibt es noch die Embedded-Control-Variante Gforth EC, die u.a. auf dem R8C läuft.

Geschwindigkeit Gforth verwendet eine einfache Methode zur Maschinencodeerzeugung und ist damit üblicherweise deutlich schneller als andere portable Systeme, auch wenn zu den schnellsten Forth-Compilern manchmal noch ein Faktor 2-3 fehlt.

Gforth ist über einen GNU-Mirror in Ihrer Nähe erhältlich, oder über `http://www.complang.tuwien.ac.at/forth/gforth/`; die Homepage von Gforth ist auf `http://www.jwdt.com/~paysan/gforth.html`.

Was ist neu in Gforth 0.7.0?

Die wichtigste Neuerung ist das neue **C-Interface** libcc. Es erlaubt, portable Deklartionen von C-Funktionen zu schreiben und diese C-Funktionen dann über Forth-Wörter aufzurufen. Die Schwierigkeit hierbei ist, dass die gleiche Funktion auf verschiedenen Plattformen verschiedene Typen haben kann. Libcc gleicht das aus, indem es einfach die `.h`-Dateien verarbeitet, in denen die Deklarationen mit den richtigen Typen für die jeweilige Plattform stehen, und indem die Argument-Typen des dazugehörigen Forth-Wortes explizit deklariert werden. Libcc konvertiert dann, wenn nötig, zwischen den Forth-Typen und den C-Typen, und zwar so:

Libcc erzeugt eine Adapter-Funktion in C, die die Argumente von den Forth-Stacks liest, dann die C-Funktion aufruft und am Schluss das Ergebnis wieder auf den Forth-Stack schreibt. Diese Adapterfunktion wird von einem C-Compiler compiliert (der aus den `.h`-Dateien die Argumenttypen der C-Funktion kennt), und dann von libcc dynamisch in das Forth-System gelinkt. Weiters erzeugt libcc ein Forth-Wort, das diese Adapterfunktion (und dadurch die gewünschte C-Funktion) aufruft. Der Nachteil dieser Methode ist, dass sie einen C-Compiler zur Laufzeit braucht; wir arbeiten für die nächste Version daran, diesen Nachteil für die häufigsten Fälle zu eliminieren.

Ein Beispiel für die Deklarationen und Verwendung ist:

```
\ Deklaration C-Seite:
\c #include <sys/types.h>
\c #include <unistd.h>
\ Deklaration Forth-Seite:
c-function dlseek lseek n d n -- d
\ Forth-Name: dlseek, C-Name: lseek
\ Aufruf:
fd @ 10. SEEK_SET dlseek
```

Eine weitere große Neuerung ist die Unterstützung für UTF-8 (Unicode für Zeichensätze aller möglichen Sprachen) über die **Xchars**-Wörter (siehe VD 1/2006, S. 19). Diese Zeichen können für alles verwendet werden: Für Daten, in String-Konstanten, und sogar für Wortnamen. Gforth behandelt übrigens bei ASCII-Zeichen in Wortnamen Groß- und Kleinbuchstaben als gleich, aber bei anderen Zeichen wird zwischen Groß- und Kleinbuchstaben unterschieden.

Bei den **Zahlen-Präfixen** gab es einige kleine, aber praktische Neuerungen: Leute, die mit C-basierten Werkzeugen arbeiten, werden `0x` als Hex-Präfix zu schätzen wissen (z.B. `0xff`). Weitere Änderungen in diesem Bereich kommen von Forth200x: Es gibt jetzt `#` als Dezimal-Präfix und man kann jetzt auch das Vorzeichen nach dem Präfix schreiben, z.B. `#-10`. Und schließlich kann man Zeichenkonstanten jetzt auch mit abschließendem Hochkomma schreiben: `'a'`. Dafür akzeptiert Gforth keine Mehrzeichen-Konstante wie `'ab` mehr.

Bei einem portablen System wie Gforth sind natürlich die **Ports** wichtig. Gforth enthält jetzt bessere Unterstützung für **MacOS X**. Einen großen Beitrag dazu leistete die Forth-Gesellschaft, indem sie Bernd Paysan einen Mac Mini zur Verfügung stellte. Gforth dürfte unter den Forth-Systemen für MacOS X inzwischen einen großen Marktanteil haben, wohl vor allem, weil Gforth durch seine Portabilität beim Apples-Umstieg auf Intel keine Probleme hatte.

Es gibt jetzt auch einige neue Ports für Gforth EC. Die sind allerdings teilweise nur unvollständig im Gforth-Paket; bei Bedarf sollte man sich die CVS-Version[1] anschauen.

Außerdem gibt es eine bessere Unterstützung für die Architekturen AMD64, ARM, und IA-64. Diese waren für Gforth 0.6.2 zwar noch nicht berücksichtigt und getestet worden. Trotzdem lief Gforth durch seine Portabilität dort, allerdings ohne dynamische Maschinencodeerzeugung und damit langsam. Jetzt unterstützt Gforth auch

[1] `http://www.complang.tuwien.ac.at/forth/gforth/cvs-public/`

auf diesen Architekturen die dynamische Maschinencodeerzeugung, und außerdem explizite Registerbelegung für weitere Geschwindigkeitsvorteile.

In mehreren Bereichen gibt es eine **bessere Fehlerbehandlung**: `Defer`ed words produzieren jetzt eine Warnung, wenn sie ausgeführt werden, ohne vorher initialisiert zu werden. Fehler bei der ganzzahligen Division (z.B. Division durch 0) werden jetzt auf allen Plattformen erkannt (auch ohne Hardware-Unterstützung), allerdings nur durch `gforth`, nicht `gforth-fast`.

Außerdem gibt es bessere Möglichkeiten zur Sicherheit bei Exceptions, z.B. durch Unterbrechungen: Wörter wie `outfile-execute`, `infile-execute`, und `base-execute` ermöglichen die zeitweilige Änderung von globalen Systemvariablen, so dass sie bei einer Exception oder Unterbrechung sicher wieder zurückgesetzt werden. Ähnliches kann man mit einem überarbeiteten `try`...`endtry`-Konstrukt selbst bauen.

Es gibt auch sonst eine Reihe **neuer Wörter**, z.B. für Speicherzugriffe auf ganzzahlige Werte mit 16 und 32 Bits. Oder `next-arg` und `shift-args` für den Zugriff auf Kommandozeilenargumente beim Aufruf von Gforth (nützlich für Scripts).

Maschinencode kann jetzt besser angezeigt werden: Es gibt einen Assembler/Disassembler für die PowerPC Architektur (inklusive 64-bit-Befehle). Weiters gibt es noch `disasm-gdb`, der den GNU Debugger zum Disassemblieren aufruft. Auf den Architekturen, für die es noch immer keinen in Forth geschriebenen Disassembler in Gforth gibt, ist `disasm-gdb` der Standard-Disassembler (davor `dump`). Und schließlich gibt es noch `see-code`, ein Decompiler für colon definitions, der einem den dynamisch erzeugten Maschinencode anzeigt und wie er mit den Befehlen im threaded code im Zusammenhang steht.

Im Bereich **Geschwindigkeit** hat sich auch einiges getan: Gforth verwendet ja einen C-Compiler für die Portabilität und spezielle Erweiterungen von GCC zur Leistungssteigerung. GCC bekam in der 3.x-Reihe mit jeder Version neue „Optimierungen“, die dafür sorgten, dass die dynamische Codeerzeugung von Gforth nicht mehr funktioniert hat und Gforth um mindestens den Faktor 2 langsamer wurde. Wir fanden Workarounds für diese Probleme und bauten sie in Gforth ein. Inzwischen ist mit gcc-4.4 schon eine neue gcc-Version seit Gforth 0.7.0 erschienen, und brachte keine neuen Geschwindigkeits-Probleme.

Bisher hatte die Installation von Gforth in der Standardeinstellung konservative Einstellungen gewählt, die zu deutlichen Einbußen in der Geschwindigkeit im Vergleich zu den schnellsten Einstellungen führen, aber dafür mit großer Sicherheit ein funktionierendes Forth-System bauen. Die Benutzer sollten aggressivere Konfigurationsparameter ausprobieren, um ein schnelleres System zu bekommen. Allerdings hat sich bei den Linux-Distributoren keiner diese Mühe gemacht, und daher war Gforth unter Linux relativ langsam. Jetzt probiert die Installation verschiedene Einstellungen durch, angefangen von den aggressivsten, bis sie eine findet, die funktioniert; dadurch werden die optimalen Einstellungen automatisch gefunden und das resultierende System ist dadurch deutlich schneller. Allerdings sind wir mit dieser Methode an die Grenze dessen gestoßen, was in einem Makefile sinnvoll machbar ist.

Die gemischtgenaue Division wurde neu implementiert und ist auf praktisch allen Plattformen nun deutlich schneller.

Einige Optimierungen helfen speziell bei bestimmten Architekturen: Static Stack Caching hilft vor allem auf PowerPC (32-bit und 64-bit), weil wir nur dort genügend Register verwenden können, dass es hilft. Branch target alignment beschleunigt Alphas deutlich. Die bessere Unterstützung für AMD64, ARM und IA-64 wurde schon früher erwähnt.

Daneben wurde noch eine Reihe Fehler behoben, und alle bisher verabschiedeten Forth200x-Erweiterungen eingebaut.

Minimaler Basis–Befehlssatz für ein Forth–System

Willi Stricker

Es wird ein kleiner Basis–Befehlssatz vorgestellt, der es erlaubt, alle weiteren Forth–Befehle zu erzeugen.

Er enthält möglichst wenige Befehle und keine, die durch andere Basisbefehle erzeugt werden können. Diese sollen so gestaltet sein, dass damit ein vollständiges Forth–System ohne weitere Primitives (in Assembler geschriebene Befehle) aufgebaut werden kann. Ausgeschlossen ist jedoch der Zugriff auf die Stack–Pointer und die Stacks über Fetch und Store–Befehle.

Ursprüngliches Ziel war es, Forth–Systeme für unterschiedliche Micro–Controller mit möglichst wenig Aufwand zu schaffen. Das bietet folgende Vorteile: Der Aufwand für die (Assembler–) Programmierung der Controller ist verhältnismäßig gering, die Controller sind in ihrer (Forth–)Struktur nach außen identisch und sie können mit der gleichen Entwicklungs–Software bearbeitet werden. Jetzt soll der Befehlssatz auch als Basis für einen möglichst einfach aufgebauten Forth–Prozessor dienen, der wiederum die gleiche Entwicklungs–Software benutzen kann.

Definition und Einteilung der Befehle

System–Befehle

`;S ( -> )`
Return = pop address from return stack and store to IP

`LIT ( -> data )`
load immediate data on stack

`BRANCH ( -> )`
branch to the following address

`?BRANCH ( data -> )`
branch to the following address if data equals zero else continue

Diese Befehle sind für den Benutzer normalerweise nicht sichtbar, sie werden vom Compiler zum Aufbau von Befehls–Strukturen und Immediate Daten benötigt.

Indirekter Befehls– und Unterprogramm–Aufruf

`EXECUTE ( addr -> )`
execute address (= instruction)

Parameter– und Return–Stackpointer – laden und speichern

`RP@ ( -> RP )`
get Return–Pointer

`RP! ( RP -> )`
store Return–Pointer

`SP@ ( -> SP )`
get Stack–Pointer

`SP! ( SP -> )`
store Stack–Pointer

Diese Befehle werden bei Benutzung mehrerer Stacks etwa bei Interrupt–Programmen oder für Multitasking–Systeme benötigt.

Return–Stack–Befehle

`R> ( data -> )`
pop data from return stack

`>R ( -> data )`
push data to return stack

Parameter–Stack–Befehle

`DROP ( data -> )`
drop data from stack

`PICK ( position -> data )`
load data from relative stack position

`-PICK ( data position -> )`
store data on relative stack position

Die Befehle `PICK` und `-PICK` werden für den wahlfreien Zugriff zum Parameter–Stack gebraucht.

Logik–Befehle

`INVERT ( data -> [not data] )`
bitwise NOT

`AND ( data0 data1 -> [data0 and data1] )`
bitwise AND

`OR ( data0 data1 -> [data0 or data1] )`
bitwise OR

Arithmetische Befehle

`+C ( data0 data1 -> [data0+data1] carry )`
add data with sum and carry (lsb)

`U2/C ( data -> carry [data/2] )`
shift right one bit with result and carry (msb)

Die Erweiterung der üblichen Befehle `+` und `U2/` durch das Carry–Bit (als Wort) ist notwendig für die Möglichkeit zur Erweiterung der Befehle auf doppelte Genauigkeit, da Forth kein Flag–Register kennt.

Speicher–Zugriff

`@ ( addr -> data )`
fetch data from memory address

`! ( data addr -> )`
store data on memory address

Byte–Befehle

`CSWAP ( byte0:byte1 -> byte1:byte0 )`
swap bytes in data

`C@ ( addr -> byte:0 )`
fetch byte from address (upper byte = 0)

`C! ( byte:0 addr -> )` store byte to address (upper byte is discarded)

Diese Befehle sind im Prinzip substituierbar, machen dabei aber besondere Probleme, deswegen werden sie hier aufgeführt.

Interrupt–Steuerung

`DISINT ( -> )`
disable interrupts

`ENINT ( -> )`
enable interrupts

Forth kennt als Programmiersprache zwar keine Interrupts und damit keine Interrupt–Befehle, Micro–Controller aber haben immer welche. Deswegen sind diese Befehle notwendig.

Es zeigt sich, dass mit Hilfe dieser geringen Anzahl von 26 Befehlen ein vollständiges Forth–System aufgebaut werden kann. Dazu wird im Folgenden ein Forth–Kernel vorgeschlagen, der mit Hilfe des minimalen Befehlssatzes programmiert wird.

Die Reihenfolge der Definitionen ist dabei wichtig, da später definierte Befehle vorangegangene benutzen. Der Befehlssatz ist auf minimalen Speicherbedarf optimiert. Er erhebt aber keinen Anspruch auf Vollständigkeit!

Definition eines Forth–Kernels bei Benutzung des Minimal–Befehlssatzes

```
1   : DUP ( data -> data data ) 0 PICK ;
2
3   : SWAP ( data0 data1 -> data1 data0 ) DUP 2 PICK 1 -PICK 1 -PICK ;
4
5   : R@ ( -> data ) R> R> DUP >R SWAP >R ;
6
7   : OVER ( data0 data1 -> data0 data1 data0 ) 1 PICK ;
8
9   : ROT ( data0 data1 data2 -> data1 data2 data0 ) >R SWAP R> SWAP ;
10
11  : ?DUP ( data -> data data ) DUP IF DUP THEN ;
12
13  : + ( data0 data1 -> [data0+data1] ) +C DROP ;
14
15  : U2/ ( data -> [data/2] ) U2/C SWAP DROP ;
16
17  : +! ( data addr -> ) DUP @ ROT + SWAP ! ;
18
19  : XOR ( data0 data1 -> [data0 xor data1] ) OVER OVER INVERT AND >R
20      SWAP INVERT AND R> OR ;
21
22  : D+ ( data0_l data0_h data1_l data1_h ->
23      [data0+data1]_l [data0+data1]_h ) >R SWAP >R +C R> + R> + ;
24
25  : 1+ ( data -> [data+1] ) 1 + ;
26
27  : NEGATE ( data -> [-data] ) INVERT 1+ ;
28
29  : DNEGATE ( data_l data_h -> [-data]_l [-data]_h )
30      SWAP INVERT SWAP INVERT 1 0 D+ ;
31
32  : - ( data0 data1 -> [data0-data1] ) NEGATE + ;
33
34  : D- ( data0_l data0_h data1_l data1_h -> [data0-data1]_l [data0-data1]_h )
35      DNEGATE D+ ;
36
37  : 1- (data -> [data-1] ) 1 - ;
38
39  : 0= ( data -> flag ) IF 0 ELSE -1 THEN ;
40
41  : 0< ( data -> flag ) [ HEX ] 8000 AND 0= 0= [DECIMAL] ;
42
43  : 0> ( data -> flag ) DUP 0< SWAP 0= OR 0= ;
44
45  : = ( data0 data1 -> flag ) - 0= ;
46
47  : < ( data0 data1 -> flag ) - 0< ;
```

```
48
49  : > ( data0 data1 -> flag ) - 0> ;
50
51  : U< ( data0 data1 -> flag ) 0 SWAP 0 D- 0< SWAP DROP ;
52
53  : U> ( data0 data1 -> flag ) SWAP U< ;
54
55  : MIN ( data0 data1 -> [min(data0|data1)] ) OVER OVER > IF SWAP THEN
56      DROP ;
57
58  : MAX ( data0 data1 -> [max(data0|data1)] ) OVER OVER < IF SWAP THEN
59      DROP ;
60
61  : ABS ( data -> [abs data] ) DUP 0< IF NEGATE THEN ;
62
63  : DABS ( data_l data_h -> [abs data]_l [abs data]_h )
64      DUP 0< IF DNEGATE THEN  ;
65
66  : ROLL ( data position -> ) DUP 1+ PICK >R BEGIN DUP WHILE DUP PICK
67      >R DUP 1+ R> SWAP  -PICK 1- REPEAT DROP DROP R> ;
68
69  : 2* ( data -> [data*2] ) DUP + ;
70
71  : 2/ ( data -> [data/2] ) [ HEX ] DUP 8000 AND SWAP U2/ OR
72      [ DECIMAL ] ;
73
74  : LSHIFT BEGIN DUP WHILE SWAP 2* SWAP 1- REPEAT DROP ;
75
76  : RSHIFT BEGIN DUP WHILE SWAP U2/ SWAP 1- REPEAT DROP ;
77
78  : S>D ( data -> data_l data_h ) [ HEX ] DUP 8000 AND IF -1 ELSE 0
79      THEN ;
80
81  : D>S ( data_l data_h -> data ) DUP >R DABS OVER 0< OR IF DROP R@ 0<
82      IF [ HEX ] 8000 ELSE 7FFF  [ DECIMAL ] THEN THEN R> 0< IF NEGATE
83      THEN ;
84
85  : UM* ( data0 data1 -> uprod_l uprod_h ) 0 ROT ROT 16 BEGIN >R >R DUP
86      0< >R OVER OVER D+ R> IF R@ 0 D+   THEN R> R> 1- DUP 0= UNTIL
87      DROP DROP ;
88
89  : M* ( data0 data1 -> prod_l prod_h ) OVER OVER XOR >R ABS SWAP ABS
90      UM* R> 0< IF DNEGATE THEN ;
91
92  : * ( data0 data1 -> prod ) M* D->S ;
93
94  : UM/MOD ( data1_l data1_h data2 -> umod uquot_l uquot_h ) 0 SWAP 32
95    BEGIN >R >R OVER >R  >R OVER OVER D+ R> 2* R> 0< IF 1 OR THEN DUP
96    R@ U< NOT IF R@ - ROT 1+ ROT ROT THEN R> R> 1- DUP 0= UNTIL
97    DROP DROP ROT ROT ;
98
99  : M/MOD ( data1_l data1_h data2 -> mod quot ) OVER >R OVER OVER XOR
100   >R ABS >R DABS R> UM/MOD R> 0< IF DNEGATE THEN D>S SWAP R> 0< IF
101   NEGATE THEN SWAP ;
102
103 : M/ ( data1_l data1_h data2 -> quot ) M/MOD SWAP DROP ;
104
105 : /MOD ( data1 data2 -> mod quot ) >R S->D R> M/MOD ;
106
107 : / ( data1 data2 -> quot ) /MOD SWAP DROP ;
108
109 : MOD ( data1 data2 -> mod ) /MOD DROP ;
110
111 : */MOD ( data1 data2 data3 -> mod [data1*data2/data3] ) >R M* R> M/MOD ;
112
113 : */ ( data1 data2 data3 -> [data1*data2/data3] ) */MOD SWAP DROP ;
```

Hinweis: Der Befehl `UM/MOD` ist anders als in Forth üblich definiert: Der Quotient ist vom Typ double. Das Ergebnis ist damit immer eindeutig.

Kleine Helfer: Dump, modulo base und mit Lineal

Adolf Krüger, Michael Kalus

Das Wort dump listet Werte aus dem RAM auf. Beginnend bei der Startadresse holt es aufsteigend Byte für Byte aus dem Speicher und zeigt diese zeilenweise an. Vor jeder Zeile wird die Adresse angegeben, von der die Bytes stammen. Daneben wird dieselbe Bytefolge in ASCII-Zeichen wiedergegeben.

So hat es sich eingebürgert. Nützlich finde ich die Darstellung der Werte passend zur verwendeten Zahlenbasis, wie sie in der Uservariablen base abgelegt ist. Und die Länge der Zeile ist dabei modulo base, das erleichtert die Orientierung. Ein Lineal über der Liste nummeriert die Position der Zelle. Es wird immer mindestens eine ganze Zeile angezeigt. Gewöhnlich belasse ich solche tools wie dump nicht in der Applikation, aber sie werden bei der Entwicklung benötigt und vorübergehend dazugeladen. Darum hat es sich bei mir eingebürgert, solch ein tool am Stück zu codieren, um nicht Gefahr zu laufen, die einzelnen Teile zu verlieren oder inkompatible Teile zu laden. Drum hier der Code unfaktorisiert als Modul.

```
1   : dump ( addr n -- )
2     cr cell 2* spaces
3     over
4     \ print a ruler
5     base @ 0  do  i over + base @ mod cell .r  loop   2 spaces
6     base @ 0  do  i over + base @ mod    1 .r  loop   drop
7     \ dump bytes
8     0 ?do  cr dup 0 cell 2* d.r space
9            \ print line of bytes
10           base @ 0 do  i over + c@ cell .r loop
11           space
12           \ print same bytes in ascii again
13           base @ 0 do  i over + c@
14             $7F and dup $20 < if  drop $2E then emit
15           loop
16           key?  if leave then
17           base @ +
18    base  @ +loop
19    cr drop ;
```

```
decimal here 40 dump
            6   7   8   9   0   1   2   3   4   5  6789012345
17031116  240 191 240 191 139 255 255  11 240 191 p?p?..p?
17031126  254  11 254 255 130 255 248 132  42 255 ~.~.x.*
17031136  255 255 255 191 240 190 255 255 191 255 ?p>?
17031146  255 255 255   9   9   3  11 255 236  46 ....l.
 ok

hex here 40 dump
            C   D   E   F   0   1   2   3   4   5   6   7   8   9   A   B  CDEF0123456789AB
 103DFCC   F0  BF  F0  BF  8B  FF  FF   B  F0  BF  FE   B  FE  FF  82  FF p?p?..p?~.~.
 103DFDC   F8  84  2A  FF  FF  FF  FF  BF  F0  BE  FF  FF  BF  FF  FF  FF x.*?p>?
 103DFEC   FF   9   9   3   B  FF  EC  2E  2E   C  2F  F0  30  24  2F  DF ....l.../p0$/_
 103DFFC   FC  BF  F8  BF   0   0   0   0   0   0   0   0   0   0   0   0 |?x?............
 ok
```

Links

http://www.forth-ev.de/wiki/doku.php/words:dump

Levenshteins Edit–Distanz

Ulrich Hoffmann

Wie findet man Schreibfehler in seiner elektronischen CD–Sammlung oder den Mitgliederlisten der Forth–Gesellschaft? Für Texte bieten moderne Betriebssysteme bzw. Office–Pakete Rechtschreibprüfungen an, aber wie konfiguriert man die für Spezialaufgaben? Schöner wäre es, wenn man den Kern einer Rechtschreibprüfung in Forth hätte, um damit speziell zugeschnittene Überprüfer programmieren zu können. Unten findet sich die Definition der Edit–Distanz, die Wladimir Lewenstein 1965 definiert hat und die eine gute Basis für eine Spezial–Rechtschreibprüfung bilden kann.

Levenshtein-Distanz

Die Levenshtein–Distanz ist ein Maß für den Unterschied zweier Text–Strings, bei dem gezählt wird, wieviele der elementaren Zeichen–Editier–Operationen —

- Ersetzen: ein Zeichen wird durch genau ein anderes Zeichen ersetzt,
- Einfügen: ein Zeichen muss eingefügt werden und
- Löschen: ein Zeichen muss gelöscht werden —

verwendet werden müssen, um den einen String in den anderen zu überführen.

Zum Beispiel benötigt man 2 Operationen, um den Text *Forth* in den Text *Forst* zu verwandeln:

1. *Forth* → *Forsth* (s einfügen) und
2. *Forth* → *Forst* (h löschen).

Der Standard–Algorithmus für die Levenshtein–Distanz bedient sich einer $(m+1)\times(n+1)$–Matrix D (m, n sind die Längen der zu vergleichenden Strings), die nach folgender Rekursionsgleichung berechnet wird:

$$
\begin{aligned}
D_{i,0} &= i \quad \text{für alle } i \in \{0,\ldots,m\} \\
D_{0,j} &= j \quad \text{für alle } j \in \{0,\ldots,n\} \\
D_{i,j} &= \min \begin{cases} D_{i-1,j-1} & +0 & \text{gleicher Buchstabe} \\ D_{i-1,j-1} & +1 & \text{ersetzen} \\ D_{i,j-1} & +1 & \text{einfügen} \\ D_{i-1,j} & +1 & \text{löschen} \end{cases} \\
& \quad \text{für alle } i \in \{1,\ldots,m\} \text{ und } j \in \{1,\ldots,n\},
\end{aligned}
$$

wobei schließlich in $D_{m,n}$ der Wert der Levenshtein–Distanz berechnet wird.

Schreibfehlererkennung

Wie kann man nun mit der Levenshtein–Distanz Schreibfehler erkennen? Nun — man vergleicht die Worte mit einer Referenz–Menge, im einfachsten Fall mit sich selbst, und betrachtet die Worte, die eine Edit-Distanz zwischen 1 und 2 haben. Bei 1 ist möglicherweise ein Buchstabe falsch, bei 2 hat man eventuell einen Buchstabendreher. Es ist erstaunlich, wie viele Fehler sich schon mit dieser einfachen Methode finden lassen...

Links

[1] Levenshtein-Distanz, Wikipedia, `http://de.wikipedia.org/wiki/Levenshtein-Distanz`

[2] Vladimir I. Levenshtein: Binary codes capable of correcting deletions, insertions, and reversals. In: Doklady Akademii Nauk SSSR, 163(4) S. 845–848, 1965 (Russisch). Englische Übersetzung in: Soviet Physics Doklady, 10(8) S. 707–710, 1966.

Listing

```
1  \ Levenshtein's Edit Distance                                  uho 2009-10-04
2  \
3  \ This program defines the word
4  \
5  \          LEVENSHTEIN ( c-addr1 len1 c-addr2 len2 -- u )
6  \
7  \ which calculates the edit distance u of the two given strings.
8  \
9  \ This is an ANS Forth Program with environmental dependencies
10 \
11 \    - Requiring ] [ THEN SWAP ROT R@ R> LOOP LITERAL J IMMEDIATE
12 \        IF I ELSE DUP DROP DOES> DO CREATE CR CHARS CELLS CELL+ C@
13 \        @ >R = ; : 2DUP 2DROP 1- 1+ . - , + * ! S" (
14 \      from the Core word set.
15 \   - Requiring MARKER .( \ from the Core Extensions word set.
16 \   - Requiring THROW from the Exception word set.
17 \   - Requiring S" ( from the File Access word set.
18 \   - Requiring FREE ALLOCATE from the Memory-Allocation word set.
```

```
\   - Requiring [THEN] [IF] [ELSE] from the Programming-Tools Extensions word
\     set.
\
\ Required program documentation
\   - Environmental dependencies
\     * This program has no known environmental dependencies.
\     * Using lower case for standard definition names.
\
\   - Other program documentation
\     * After loading this program, a Standard System still exists.

: Matrix ( -- )
     Create 0 ( 'buffer ) , 0 ( rowsize ) , 0 ( colsize ) ,
   Does> ( -- a-addr ) ;

: >buffer ( 'matrix -- ''buffer ) ; immediate
: >rowsize ( 'matrix -- 'rowsize ) cell+ ;
: >colsize ( 'matrix -- 'colsize ) [ 2 cells ] Literal + ;

: [][] ( row col 'matrix -- a-addr ) >r
    r@ >colsize @ *
    swap + cells
    r> >buffer @ + ;

: allocate-matrix ( colsize rowsize 'matrix -- )
    >r dup >r cells  swap  dup >r
    * cells allocate throw
    r> ( rowsize )
    r> ( colsize ) r@ >colsize !
                   r@ >rowsize !
    r> >buffer ! ;

: free-matrix ( 'matrix -- )
    dup >r >buffer @ free throw
    r@ >buffer off
    r@ >rowsize off
    r> >colsize off ;

: last-element ( 'matrix -- a-addr )
   dup >r >colsize @ 1-  r@ >rowsize @ 1- r> [][] ;

: init-col0 ( 'matrix -- )
    dup >rowsize @ 0
    DO  dup   0 I rot [][]   I swap !  LOOP
    drop ;

: init-row0 ( 'matrix -- )
    dup >colsize @ 0
    DO  dup   I 0 rot [][]   I swap !  LOOP
    drop ;

Matrix dist

: edit-distance ( c-addr1 c-addr2 -- )
    dist init-col0
    dist init-row0
    dist >colsize @ 1
    DO
       dist >rowsize @ 1
       DO ( c-addr1 c-addr2 )
          2dup I 1- chars + c@
          swap J 1- chars + c@  = IF 0 ELSE 1 THEN
          J 1- I 1- dist [][] @ +
          J    I 1- dist [][] @ 1+ umin
          J 1- I    dist [][] @ 1+ umin
          J I dist [][] !
```

```
85          LOOP
86        LOOP
87        2drop ;
88
89    : levenshtein ( c-addr1 len1 c-addr2 len2 -- u )
90       1+ rot 1+ dist allocate-matrix
91       edit-distance
92       dist last-element @
93       dist free-matrix ;
94
95    Marker *Test*
96
97    : Tor s" Tor" ;
98    : Tier s" Tier" ;
99    Tier Tor levenshtein dup 2 -
100
101   *Test*
102   [IF]
103     . .( Levenshtein test failed!) abort
104   [ELSE]
105     drop .( Levenshtein) cr
106   [THEN]
```

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 75, August 2009

Een zwaaistok — Willem Ouwerkerk

Was ist ein zwaaistok (wörtlich: Schwenkstock)? So fängt der Artikel von Willem an. Willems *rhetorische Antwort*: Ein Stäbchen aus Plastik mit 8+1 Leuchtdioden, das beim Hin–und–her–Schwenken einen Text oder eine Zeichnung in die Luft projiziert. Willem spricht über Hardware, Arbeitsweise und Software (mit einem ausgedehnten Forth–Programm). Ein Micro–Controller misst die Zeit, die das Hin–und–her–Schwenken in Anspruch nimmt, und schaltet die Leuchtdioden über einen Micro–Switch so ein und aus, dass sie den Text oder die Zeichnung richtig wiedergeben. Frage des Rezensenten: Für einen unregelmäßigen Schwenker (aus Fleisch und Blut) ist das Verfahren wohl nicht gedacht?

Eerste ervaringen met Arduino bordje — Ron Minke

Ron hat sich eine Arduino-Experimentierplatine des Typs *Duemilanove* (Italienisch für *2009*) angeschafft und berichtet über seine ersten Erfahrungen. Auf der Platine sitzt ein ATmega328P (mit 32 KB Flash, 1 KB EEPROM und 2 KB internem RAM). Auf der Webseite `www.arduino.cc` findet man eine Entwicklungsumgebung (mit einer C–ähnlichen Sprache). Ron würde gern Forth auf dem Arduino laufen sehen. Dazu bräuchte man das Programm AVRDude, das Maschinen–Code von der Entwicklungsumgebung zur Platine befördert. Mehr will er das nächste Mal darüber sagen. Weitere Informationen: `www.atmel.com/dyn/resources/prod_documents/doc2525.pdf`. (Rezensent: Frans van der Markt hat im Vijgeblaadje 73 (2009) in einem ähnlichen Bericht über Erfahrungen mit der Arduino–Platine auf die Existenz von AMForth für AVR–Prozessoren hingewiesen.)

Forth200x — Berichte von den Standardisierungstreffen

Anton Ertl

Forth 200X-Treffen auf der EuroForth 2007

Am Tag vor der EuroForth 2007 in Schloss Dagstuhl fand wieder einmal das Treffen des Forth-200X-Komitees statt, das am nächsten Forth-Standard arbeitet. Die Teilnehmer waren Sergey N. Baranov, Willem Botha, Federico de Ceballos, Anton Ertl, Ulrich Hoffmann, Peter Knaggs, Dagobert Michelsen, Bernd Paysan, Stephen Pelc, und Carsten Strotmann.

Der Großteil der Diskussion drehte sich natürlich um die Vorschläge für Forth-Erweiterungen, die den RfD/CfV-Prozess durchlaufen hatten (siehe @urlhttp://www.forth200x.org/rfds.html). Die folgenden Vorschläge wurden beschlossen:

Structures Diese Erweiterung erleichtert es, Datenstrukturen mit benannten Feldern zu definieren, ähnlich wie `struct`s in C.

Throw IORs Dieser Vorschlag definiert einige weitere Werte, die ein System bei bestimmten Fehlern mit `throw` werfen kann.

EKEY return values Mit dieser Erweiterung können Standard-Programme bestimmen, ob z.B. eine Cursor-Taste gedrückt wurde. Nachdem dieser Vorschlag beim Treffen 2006 noch für eine Klarstellung zurückgestellt worden war, wurde er diesmal angenommen (die Klarstellung wurde allerdings beim Treffen vor der Forth-Tagung 2009 noch einmal überarbeitet).

Number Prefixes standardisiert die Schreibweise von Zahlen o.ä. mit einem Präfix, der die Basis angibt, z.B. `$1f` für einen Hex-Wert.

Weiters wurden noch zwei Änderungsvorschläge angenommen, die den Text des Standarddokuments ändern, ohne den Inhalt signifikant zu ändern, darunter eine Änderung bei der Definition von `to`, die künftige Erweiterungen (z.B. `2value`) einfacher machen soll.

Schließlich wurde noch der Vorschlag **separate fp stack** diskutiert, der den separaten Gleitkomma-Stack zum Standard erklären soll. Dieser Vorschlag wurde bei diesem Treffen noch zurückgestellt, um eine Beschreibung der Auswirkungen in das Standard-Dokument einzuarbeiten, und wurde dann beim Treffen 2008 angenommen.

Außerdem wurden bei dem Treffen noch Vorschläge diskutiert, die noch nicht die CfV-Stufe erreicht hatten:

Enhanced locals schlagen einerseits eine bessere Syntax für locals vor, andererseits auch Erweiterungen wie lokale Buffer.

Escaped strings Hier geht es um Möglichkeiten, Sonderzeichen oder `"` in String-Literale einzubauen.

Synonyms erlauben es, einen neuen Namen für ein existierendes Wort zu definieren.

Extended characters Die xchar-Erweiterung erlaubt die Verwendung von Zeichenkodierungen wie UTF-8 (siehe VD 1/2006, S. 19).

2VALUE, FVALUE Ähnlich wie VALUE für andere Datentypen.

Directories Wie referenziert man eine andere Datei innerhalb des gleichen Forth-Programms?

Weiters wurden noch mögliche zukünftige RfDs präsentiert.

Zusätzlich gab's noch Diskussionen über das vorläufige Standard-Dokument und über diverse organisatorische Themen.

Wer's genauer wissen will, findet noch die Minutes von Peter Knaggs und meinen englischen Bericht auf `http://www.forth200x.org/forth200x.html`.

Forth-200X-Treffen auf der EuroForth 2008

Am Tag vor der EuroForth 2008 in Wien fand wieder einmal das Treffen des Forth-200X-Komitees statt, das am nächsten Forth-Standard arbeitet. Die Teilnehmer waren Willem Botha, Federico de Ceballos, Anton Ertl, Andrew Haley, Ulrich Hoffmann, Peter Knaggs, Bernd Paysan, Stephen Pelc, Bill Stoddart, Willi Stricker, Carsten Strotmann und Leon Wagner.

Diesmal stand nur ein Erweiterungs-Vorschlag zur Abstimmung an: **Separate FP stack** zur Standardisierung eines separaten Gleitkomma-Stacks, der schon beim letzten Treffen diskutiert worden war. Nach einigen weiteren Verfeinerungen wurde dieser Vorschlag endlich angenommen.

Im letzten Standard (Forth-94) wurden einige Wörter als veraltet (*obsolescent*) deklariert und sollten im kommenden Standard entfernt werden. Wir beschlossen, diese Wörter noch in einem Anhang zu dokumentieren.

Es gab wieder Diskussionen zu den gleichen Vorschlägen im RfD-Stadium wie im Vorjahr. Die meisten wurden inzwischen als reif für einen CfV befunden, aber bei den Locals ergab sich während des Treffens eine Meinungsänderung des Proponenten Stephen Pelc, der zusammen mit Leon Wagner den Vorschlag noch einmal von Grund auf überarbeiten will.

Weiters gab es noch Diskussionen über größere Projekte wie die Internationalisierung und Cross-Compilation.

Das nächste Treffen mit Abstimmungen fand am 2.-4.9.2009 unmittelbar vor der EuroForth 2009 in Exeter in England statt. Englischsprachige Notizen dazu finden sich ebenso unter `http://www.forth200x.org/forth200x.html`.

Bootmanager und FAT–Reparatur: Fünfter Fort(h)schritt (HD–Parameter)

Fred Behringer

Im dritten Teil dieser Serie (VD–Heft 1/2009) habe ich Forth–Worte vorgeschlagen, mit denen man den vollen und schnellen Zugriff auch auf solche Festplatten–Sektoren bekommt, die man wegen ihrer hohen Adresse nicht mehr über die CHS–Adressierung erreichen kann: Es ging um die LBA–Adressierung für Sektoren, die jenseits der 8–GB–(DOS–)Grenze liegen [FB]. Diesmal will ich über die zugehörigen Informationen im MBR sprechen, die in gewisser Weise redundant sind, und darüber, wie man die Redundanz eventuell dazu ausnutzen kann, mit Forth–Mitteln einen zerstörten MBR wiederherzustellen. Einige Festplatten–Parameter erhält man über den erweiterten Interrupt 13h mit dessen Funktion 48h, andere über den MBR, die EBRs und generell über die Bootsektoren der primären Partitionen und der logischen Laufwerke der erweiterten Partition. Ich spreche hauptsächlich über FAT16 und ich bemühe mich, die genannten Informationen per Forth bereitzustellen. Es wird die freie Auswahl der gewünschten Festplatte zugelassen und es wurden Fehlerberücksichtigungs–Mechanismen eingebaut. Die hier entwickelten Programme stützen sich zum Teil auf die Programme in den Teilen 1 bis 3 dieser Artikelserie. Die Programme aus Teil 4 werden nicht benötigt.

Beweggründe

Versucht man, den *kostenlosen* Partition–Table–Doctor von `www.partitiontool.com` [PM] herunterzuladen, tut man sich schon beim Durchhangeln auf der HTML–Page schwer. Beim einfachen Herunterladen der kostenlosen Teile, wobei vom Anwenden noch gar keine Rede ist. Mit Programmen anderer Quellen zum Reparieren einer zerstörten Partitionstabelle ist es ähnlich. Und was für ein Aufwand da getrieben wird! Nur, um herauszubekommen, welche Hexzahl ich wohin zu schreiben habe, um den Bootmanager über die Partitionstabelle des MBRs, des Master–Boot–Records, zum Bootsektor meines Windows–ME–Systems irgendwo an *höherer* Stelle auf der Festplatte zu leiten. Unerträglich, wenn man sich die Mühe gemacht hat, hinter die Kulissen zu schauen, und feststellen muss, dass ein paar Forth–Worte genügen, die Dinge ins Lot zu bringen!

Natürlich besteht das eigentliche Problem (bei beschädigtem MBR) nicht darin herauszubekommen, wo in der Partitionstabelle des MBRs der herauszufindende Eintrag zu machen ist, sondern welcher Eintrag, wenn dieser, beabsichtigt oder unbeabsichtigt, unzulässig verändert wurde. Ich werde versuchen, Mittel dazu bereitzustellen. Auf dem Weg dorthin hat mich das FreeWare–Programm CWDSKEDT (Christoph–Walter–Disk–Editor) recht gut geleitet [CW]. Es ist für DOS konzipiert, arbeitet aber auch unter Windows bis zu ME. Bis zu einem gewissen Grad würde es sicher reichen, sich eine Sammlung von solchen Fix–und–Fertig–Programmen zurechtzulegen. Aber nur bis zu einem gewissen Grad! Jeden Augenblick stößt man an einen Punkt, an welchem man nicht weiterkommt, weil der eigentliche Durchblick fehlt. Anders bei einem Programmteil, den ich in Forth selbst geformt habe. Da finde ich mich leicht zurecht! Und was noch nicht geht, wird ergänzt! Forth macht's möglich.

Hintergrund

Im Wikipedia [WI] findet man folgenden Eintrag (nur Auszug relevanter Teile):

Wiki–Anfang 1

> Aufbau der Partitionstabelle bei IBM–PC–kompatiblen Computern
>
> Auf Festplatten für IBM–PC–kompatible Computer wird üblicherweise der unten beschriebene Aufbau der Partitionstabelle verwendet. Auf zum PC kompatiblen Speichermedien (z.B. Compact Flash Karten, USB–Stick, Zip–Diskette) ist diese Art von Partitionierung möglich, aber nicht nötig. Auf Speichermedien anderer Systeme (z.B. Workstations, Apple Macintosh, Amiga mit RigidDiskBlock) werden andere Partitionstabellen verwendet, die hier nicht beschrieben werden.

Wiki–Ende 1

Ich halte mich an die eben genannten Voraussetzungen. Als Beispiel darf ich die (im Master–Boot–Record, dem MBR, enthaltene) Partitionstabelle meines Experimentier–Computers (AMD–K6–2/500, 384–MB–RAM, 20–GB–HD) angeben (vergleiche Teil 1 dieser Artikelserie [FB] im VD–Heft 3/2008). Man bekommt den MBR über getmbr showsectbuf, und die eigentliche Partitionstabelle erreicht man detailliert aufbereitet über showparttab . Hier der interessierende Auszug:

```
Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

 0000  eb 48 90 10 8e d0 bc 00 b0 b8 00 00 8e d8 8e c0
 0010  fb be 00 7c bf 00 06 b9 00 02 f3 a4 ea 21 06 00
                           .  .  .

 0190  61 64 00 20 45 72 72 6f 72 00 bb 01 00 b4 0e cd
 01a0  10 ac 3c 00 75 f4 c3 00 00 00 00 00 00 00 00 00
 01b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
 01c0  01 00 16 fe 3f 3f 3f 00 00 00 01 b0 0f 00 00 00
 01d0  01 80 05 fe ff ff 80 60 1f 00 80 a3 db 00 00 00
 01e0  01 40 16 fe 3f 7f 40 b0 0f 00 40 b0 0f 00 80 00
 01f0  c1 ff 0c fe ff ff ad 9e b2 01 4e 01 b0 00 55 aa
```

Wiki–Anfang 2

Wie unten erläutert, gibt es primäre und erweiterte Partitionstabellen. Ihr Aufbau ist identisch. Die Tabelle fängt bei Offset 1beh des jeweiligen Sektors an und hat maximal 4 Einträge à 16 Bytes, sie ist also 64 Byte groß. Daran anschließend folgt die *Magic–Number*, die aus den beiden Bytes 55h und aah an den letzten zwei Bytestellen des Sektors besteht.

Die (16 Byte großen) einzelnen Zeilen haben folgende Bedeutung:

```
Speicherplatz-Adresse (hexadezimal und relativ zum Zeilenanfang)
 | Größe (Bytes)
 |   | Inhalt
 |   |  |
00   1  Partition bootbar (80 = ja, 00 = nein,
                          81 = zweite Festplatte usw.)
01   3  CHS-Eintrag des ersten Sektors
        (CHS = Cylinder,Head,Sector)
04   1  Partitionstyp (06 = FAT16, 05 = erweitert,
                     0C = FAT32, 16 = 06 versteckt usw.)
05   3  CHS-Eintrag des letzten Sektors
08   4  Startsektor (relativ zum Anfang der Festplatte,
                    oder zur erweiterten Partitionstabelle)
0C   4  Anzahl der Sektoren in der Partition
```

Unbenutzte Einträge sind mit Nullen gefüllt.

Wiki–Ende 2

Die CHS–Einträge sind wie folgt kodiert (vergleiche Teil 3 im VD–Heft 1/2009 [FB]):

Byte 0	=	H	:	HHHHHHHH
Byte 1	=	S	:	CCSSSSSS
Byte 2	=	C	:	CCCCCCCC

Die Reihenfolge der Bytes entspricht der Bezeichnung HSC, die übliche Bezeichnung lautet CHS.

Die Reihenfolge der Nummern der Bitstellen in den Bytes ist `76543210`. Für die Sektorbits in Byte 1 gilt: `543210`. Für die Zylinderbits gilt: `98 76543210`, wobei `98` die Überhang–CC–Bitstellen aus Byte 1 sind.

Die Zählung der Köpfe und die der Zylinder beginnt bei 0. Man muss also zur Nummer eines Kopfes oder eines Zylinders noch 1 hinzuzählen, um die Anzahl der beteiligten Köpfe (oder Zylinder) einschließlich des betrachteten zu erhalten. Die Zählung der Sektoren beginnt bei 1. Die Anzahl der Sektoren (im jeweiligen Kopf) ist also mit der Nummer des (gerade noch enthaltenen) Sektors identisch.

Man darf somit bei den Wikipedia–Angaben (siehe gleich) nicht überrascht sein, wenn man `1023`, `254`, `63` liest, und nicht etwa `1024`, `255`, `63` (Angaben dezimal).

Überprüfung des CHS–Eintrags der Windows–ME–Partition bei meinem Experimentier–Computer: Es sind ja eigentlich zwei CHS–Einträge, Anfangs– und Endwerte. Sie taugen nicht zur Festlegung der Adresse des ME–Bootsektors, da dessen Adress–Wert zu groß ist (und nur per LBA erfasst werden kann): HSC–Eintrag, erster Kopf, erster Sektor, erster Zylinder: `00 01 3ff`, dezimal: `000 01 1023` HSC–Eintrag, letzter Kopf, letzter Sektor, letzter Zylinder: `fe 3f 3ff`, dezimal: `254 63 1023`

Man beachte, dass bei der Werteangabe (sowohl bei den Anfangswerten wie auch bei den Endwerten) die höchsten beiden Bits von Byte 1 dem höchsten Bit von Byte 2 vorangesetzt werden müssen: 6 Sektor–Bits, 10 Zylinderbits.

Irgendwie scheint mir dieses Ergebnis nur bedingt mit dem bei Wikipedia Gesagten (siehe gleich) zu harmonieren. Besser wäre wohl ein Vergleich mit den Zifferrädern des Kilometerstand–Zählers im Auto bei Überlauf gewesen. Ein Koffer mit Zahlenschloss wäre für den Vergleich auch nicht schlecht. Beim ersten Kopf und beim ersten Sektor der CHS–Zählung liegt bei Überlauf kein Grund vor, in der Partitionstabelle den möglichen Höchststand anzuzeigen.

Wiki–Anfang 3

Da bei modernen Festplatten die Adressierung durch CHS (1024 Zylinder * 255 Heads * 63 Sektoren * 512 Bytes = ca. 8 GB) nicht ausreicht, wird ab dieser Größe immer 1023, 254, 63 für die CHS–Werte angegeben und Position und Größe werden dann ausschließlich durch die Sektorangaben festgelegt. Somit ergibt sich auch die maximale Größe von 2 TB für eine Partition und 4 TB für eine Festplatte (2^{32} * 512 Bytes = ca. 2 TB). Die Zylinder und Köpfe werden beginnend bei 0 und die Sektoren beginnend bei 1 gezählt.

Wiki–Ende 3

Soweit die Angaben bei Wikipedia. Es folgen ein paar ergänzende Erläuterungen von mir:

Bei der CHS–Adressierung werden die Sektoren tatsächlich ab 1 nummeriert, bei der LBA–Adressierung aber ab 0 !

Eine besondere Eigenschaft des Verhältnisses *Anzahl der Vorgänger zur betreffenden Nummer* ist bemerkenswert: Die (Ordnungs–)Zahl 5 beim LBA–Sektor Nummer 5 ist gleichbedeutend mit der (An–)Zahl der bis dato schon durchlaufenen LBA–Sektoren `0 1 2 3 4`. (5 lässt sich bei dieser Erklärung am Beispiel natürlich durch eine beliebige andere natürliche Zahl n ersetzen.)

Beispiel: Die Eintragung ad 9e b2 01 in der obigen Partitions–Tabelle (im MBR meiner Experimentier–Anlage), also die (Hex–)Zahl `01b29ead`, gibt die Anzahl der schon durchlaufenen (LBA–)Sektoren wieder (dezimal geschrieben, `28483245`). Genau so, nämlich als *# preceding sectors*, wird das auch an verschiedenen Stellen in der Literatur beschrieben. Aber eben diese Zahl kann auch als (LBA–)Sektornummer des Anfangs der betreffenden Partition (hier von Windows ME) aufgefasst werden. Mit anderen Worten: Die Partitionstabelle im MBR trägt schon *von Hause aus* alle Informationen in sich, die nötig sind, um mit einem Bootmanager an die zu bootende Partition zu gelangen — die verwendete Software braucht das nur gebührend zu berücksichtigen, und sich nicht etwa auf die CHS–Zählung zu verlassen. Aber Achtung: Bei primären Partitionen beginnt die Zählung der LBA–Sektoren am Anfang der Platte, bei den logischen Laufwerken der erweiterten Partition

dagegen am Anfang der erweiterten Partition (nicht am Plattenanfang, aber auch nicht etwa beim Anfangssektor (dem EBR) des jeweiligen logischen Laufwerks).

Festplatten–Organisation

Es gibt pro Festplatte nur einen einzigen MBR. Das ist immer der LBA–Sektor mit der Nummer 0. Alle Primärpartitionen beginnen mit ihrem Bootsektor. (Der MBR gehört zur gesamten Festplatte, nicht etwa nur zur physikalisch ersten Partition.) Vier Partitionen sind (pro Festplatte) möglich. Eine und nur eine dieser Partitionen kann als erweiterte Partition eingerichtet werden. Jedes logische Laufwerk der erweiterten Partition beginnt mit einem MBR–ähnlichen Sektor, dem EBR (Extended–Boot–Record). Die EBRs bilden eine verkettete Liste, über die man sich zum interessierenden logischen Laufwerk durchhangeln kann. Ein Zeiger im EBR ist entweder auf lauter Nullen gesetzt oder weist auf das eventuell vorhandene nächste logische Laufwerk. Der Bootsektor des betreffenden logischen Laufwerks liegt um so viele Sektoren vom EBR entfernt, wie der EBR–Eintrag bei Offset `1c6` angibt. Normalerweise sind das `63d` Sektoren, also die Länge eines *Kopfes*.

Little–Endian bei 64 Bit?

Es gilt bei INTEL–Maschinen (PC–Kompatible gehören dazu) für im Speicher abgelegte Mehr–Byte–Werte die Little–Endian–Vereinbarung: Der RAM–Eintrag `4e 01 b0 00`, die Anzahl der in meiner Windows–ME–Partition (siehe obige Tabelle) enthaltenen Sektoren, ist als 00b0014e zu lesen! Ich will mal schnell in meinem Forth–System die Adresse `7000` (alle Angaben hexadezimal) als freie RAM–Stelle willkürlich wählen. `1234 5678 7000 2!` liefert an der Adresse `7000` die Bytes `78 56 34 12`, also streng little–endian. Man überzeugt sich davon per `7000 20 dump`. Versucht man aber, den Wert per `7000 2@ ud.` wieder hereinzuholen, so erhält man `56781234` als Anzeige auf dem Bildschirm. Ähnliches Durcheinander, diesmal im RAM, würde bei Eingabe von `12345678. 7000 2!` auftreten. Bei doppeltgenauen Werten werden oberer und unterer Anteil auf dem Datenstack vertauscht. Der (nach unten wachsende) Stack ist nun aber auch nichts weiter als ein bestimmter RAM–Bereich (2Variablen fallen ebenfalls unter diese Betrachtung). Geht man an den Stack per `1234 5678 sp@ 20 dump` heran, dann bekommt man (für den Stack–RAM–Bereich) wieder die Little–Endian–Bytefolge `78 56 34 12`.

Im vorliegenden Artikel tritt bei dem Forth–Wort `get#sects-hd#` (siehe Listing) das Problem auf, einen 64–Bit–Wert aus einem Puffer im RAM auf den Stack zu holen und auf dem Bildschirm darzustellen. Über vierfachgenaue Werte scheint es in Forth keine Empfehlungen zu geben. Soll ich vierfachgenaue Werte aus zwei zweifachgenauen zusammensetzen und ähnliche Vertauschungen auf dem Stack vornehmen wie bei den zweifachgenauen Werten (den Punktzahlen)? Ich trete die Flucht nach vorn an und verlange zum Beispiel, dass die Zahl 123456789abcdef0 in hexadezimalen Vierergruppen als `1234 5678 9abc def0` (z.B. über die Tastatur) auf den Stack gegeben wird. Sie liegt dann als strenge Little–Endian–Bytefolge im Stack–RAM–Bereich und zwischen Stack, 4Variablen und RAM besteht kein Unterschied mehr. Das Darstellungssystem und die von mir dafür vorgeschlagenen Forthworte `q@ q! q.` lassen sich leicht auf beliebiggenaue Werte ausdehnen. (Man denke an 80–Bit–Zahlen aus dem Zwischenregister der 80486–FPU, also an fünffachgenaue Werte.)

USB–Sticks als Festplatte

Was die Kennzahl `80` für *bootbar* (siehe Tabelle) betrifft, habe ich Experimente mit einem USB–Stick angestellt, und zwar unter Windows ME, da ich hier immer noch den vollen Zugriff auf meine Maschine einerseits, aber auch die USB–2.0–Unterstützung andererseits habe. Den Stick hatte ich über PARTED von einer Live–CD auf FAT16 mit mehreren Partitionen eingerichtet. Ergebnis: Wenn ich in den bisherigen Forth–Programmen der vorliegenden Artikelserie, dort, wo es sinnvoll ist, 81 statt 80 schreibe, dann kann ich alle bisher besprochenen Operationen auch am USB–Stick durchführen. Das gibt mir den Anschluss an den sehr interessanten Artikel *Forth am Stil* von Carsten Strotmann aus dem VD–Heft 2/2006 [CS]. Motto: Ich will nicht immer zeitaufwändig solange im Internet herumsuchen müssen, bis ich ein Programm finde, das das tut, was ich gerade vorhabe (was ja doch nie klappt), nein, ich möchte die Dinge selber machen. In Forth! Nicht, weil Forth die beste aller Sprachen ist — aber weil Forth mir den leichtesten und schnellsten Zugang bietet. Und ich möchte dabei lernen.

Danksagung und Vorschau

Viele wichtige Informationen zur vorliegenden Untersuchung habe ich in den MS–Unterlagen [MS] gefunden. Ich bin Johann Sturm vom Vaterstettener Senioren–Computer–Kreis für seine Hinweise auf diese und weitere Quellen zum Thema dankbar.

Demnächst will ich mich mit den Begriffen FAT, Root–Directory, Cluster und mit der Organisation von Dateien, vor allem unter FAT16, auseinandersetzen.

Literatur und Internet–Adressen

[CS] Strotmann, Carsten:	Forth am Stil. Vierte Dimension (2006), Heft 2.
[CW] Walter, Christoph:	CWDSKEDT 2.22, Freeware, Christoph–Walter–Diskedit (1999).
[FB] Behringer, Fred:	Bootmanager und FAT–Reparatur, Teil 1 bis 4. Vierte Dimension, Hefte 3–4/2008 und 1–2/2009.
[MS] Windows 2000:	`http://www.microsoft.com/technet/prodtechnol/...`
[PM] PARTITION MASTER:	`http://www.partition-tool.com` (Easeus). CHENGDU YIWO Tech Development Co., Ltd.(2009).
[WI] Wikipedia:	`http://de.wikipedia.org/wiki/Partitionstabelle`

Listing

```
\ ******************************************************
\ *                                                    *
\ *  BOOTMA-5.FTH                                      *
\ *                                                    *
\ *  Zutaten fuer FAT-Reparatur und Bootmanager unter  *
\ *  Turbo-FORTH-83 und ZF                             *
\ *                                                    *
\ *  Fred Behringer - Forth-Gesellschaft - 01.10.2009  *
\ *                                                    *
\ ******************************************************

\ Fuer Turbo-Forth:
\ FORTH INCLUDE BOOTMA-1.FTH INCLUDE ... Hierbei ist die erste Include-Datei
\ die aus Heft 3/2008, die zweite die aus Heft 4/2008 usw. Hier - bis zum
\ allerletzten INCLUDE (man darf natuerlich auch alles kleinschreiben) -
\ koennen alle Eingaben 'geschachtelt' werden, ohne per [ret] 'absetzen' zu
\ muessen.

\ Fuer ZF:
\ ZF [ret] [ret]
\ FLOAD BOOTMA-1.FTH [ret] FLOAD BOOTMA-2.FTH [ret] FLOAD usw.
\ In BOOTMA-1.FTH muss vorher attributs off per '\' auskommentiert werden (in
\ ZF gibt es kein attributs). Die Schachtelung bei FLOAD geht in ZF nicht. Die
\ Endung .FTH nicht vergessen!

\ Nicht jedes DOS-System kennt den Escape-Sequenzen-Treiber ANSI.SYS. DOS 6.2
\ hat ihn. In Turbo-Forth sorgt ANSI.SYS fuer ein 'fettes' OK-Prompt. Hat man
\ kein ANSI.SYS, dann kann man auch auf ANSI.COM (siehe Teil 3 aus dem VD-Heft
\ 1/2009) auszuweichen: ANSI.COM von Michael J. Mefford aus dem PC Magazine des
\ Jahres 1988 laesst sich im Internet (z.B. per Google) leicht beschaffen und
\ funktioniert prima.

\ Fuer ZF sind die folgenden Worte leider nicht verwendbar (es gibt dort kein
\ $execute): sect>bak bak>sect mbr>bak bak>mbr mbr>bak-hd# bak>mbr-hd#. Zum
\ Ausprobieren habe ich sie zeitweilig herausgenommen: Geht prima! Ueberprueft
\ habe ich nur stichprobenartig (Aufgabe fuer den geneigten Leser). Der gesamte
\ vorliegende Teil 5 kommt andererseits mit den Teilen 1 bis 3 als Basis aus.


\ Glossar
\ -------

\ #... = Anzahl..., #sect = LBA-Sektoranzahl
\ ...# = ...Nummer, sect# = LBA-Sektornummer, hd# = Festplatten-Nr. (80,81,...)
\ ( d#lba -- ) = Beispiel fuer einen doppeltgenauen Wert im Stack-Kommentar
\ tmp1 (--ad) Variable fuer kurzzeitige Speicherung von (Stack-)Parametern
\ tmp2 (--ad) dito
```

```
48
49  \ Zur Abgrenzung zu aehnlichen Forth-Worten aus den bisherigen Teilen der Serie
50  \ enden die Namen aller Worte, die eine Auswahl der Festplatte zulassen, ab dem
51  \ vorliegenden Artikel in -hd# . Herkoemmliche Festplatten enthalten bewegliche
52  \ Teile. Elektronik, die auf bewegliche Teile warten muss, ist fehleranfaellig.
53  \ Also habe ich bei allen Worten, deren Namen auf -hd# enden, Fehlerabfragen
54  \ ( -- fl ) eingebaut. Bei Intel-maschinennahen Worten heisst fl=0, dass kein
55  \ Fehler vorliegt, bei High-Level-Forth-Worten (Vergleiche, Abfragen) kann fl=0
56  \ 'falsch' ('nicht zutreffend') bedeuten. Besonders geachtet habe ich darauf,
57  \ auch dann noch ein sauberes fl<>0 zu bekommen, wenn bei interaktiven Eingaben
58  \ das ( hd# -- ) vergessen wird und der Stack dabei leer ist.
59
60  \ sect>kb (dsects -- dkb) Sektoranzahl nach Kilobyte. dsects doppeltgenau,
61  \ kb>sect (dkb -- dsects) Kilobyte nach Sektoranzahl. dkb auch.
62
63  \ (getmbr-hd#) (hd# -- fl) Maschinenprogrammkern fuer getmbr-hd#
64  \ getmbr-hd# (hd# -- fl)  Wie getmbr,  aber Festplatte hd# beliebig. fl=0: OK
65  \ (putmbr-hd#) (hd# -- fl) Maschinenprogrammkern fuer putmbr-hd#
66  \ putmbr-hd# (hd# -- fl)  Wie putmbr,  aber Festplatte hd# beliebig. fl=0: OK
67
68  \ mbr>bak-hd# (hd# -- fl) Wie mbr>bak, aber Festplatte hd# waehlbar. fl=0: OK
69  \ bak>mbr-hd# (hd# -- fl) Wie bak>mbr, aber Festplatte hd# waehlbar. fl=0: OK
70
71  \ testpart# (part# -- part#/1) part#=1 und raus, wenn nicht part# aus [1..4].
72
73  \ getLBAsect# (part# -- dlba#) LBA-Anfangs-Sektor  der Part. part#  aus sectbuf
74  \ getLBA#sect (part# -- d#lba) Anzahl LBA-Sektoren der Part. part#  aus sectbuf
75  \ putLBAsect# (dlba# part# --) LBA-Anfangs-Sektor  der Part. part# nach sectbuf
76  \ putLBA#sect (d#lba part# --) Anzahl LBA-Sektoren der Part. part# nach sectbuf
77
78  \ gettyp (part# -- c) Holt den Dateisystem-Typus aus part# in sectbuf
79  \ puttyp (c part# --) Legt den Dateisystem-Typus nach part# in sectbuf
80  \ putfat16 (part# --) 06 in das untere Nibble an Offset 4 von part# in sectbuf
81  \ putfat32 (part# --) 0c in das untere Nibble an Offset 4 von part# in sectbuf
82  \ putntfs  (part# --) 07 in das untere Nibble an Offset 4 von part# in sectbuf
83  \ putlin83 (part# --) Legt 83  in das Byte mit   Offset 4 von part# in sectbuf
84  \ putlin82 (part# --) Legt 82  in das Byte mit   Offset 4 von part# in sectbuf
85  \ put55-aa (--)       Legt die Magic Number ans Ende von sectbuf
86
87  \ testfat16 (part# -- fl) 06 im unt. Nibble an part#/Offs.4 in sectbuf? fl=0: No
88  \ testfat32 (part# -- fl) 0c im unt. Nibble an part#/Offs.4 in sectbuf? fl=0: No
89  \ testntfs  (part# -- fl) 07 im unt. Nibble an part#/Offs.4 in sectbuf? fl=0: No
90  \ testlin83 (part# -- fl) 83 an Byte mit Offset 4 v. part#  in sectbuf? fl=0: No
91  \ testlin82 (part# -- fl) 82 an Byte mit Offset 4 v. part#  in sectbuf? fl=0: No
92  \ test55-aa (-- fl)       55 aa am Ende von sectbuf (Offset 1fe/1ff)?   fl=0: No
93
94  \ drvparam  (--ad) Konst., Adresse des Parameterpuffers (1 Sektor = 512d Bytes)
95  \ showdrvparam    (--) Zeigt den gesamten Sektor drvparam am Bildschirm an
96  \ showdrvparam100 (--) Zeigt die ersten 100h Bytes daraus am Bildschirm an
97
98  \ (getdrvparam-hd#) (hd# -- fl) Maschinenprogrammkern fuer getdrvparam-hd#
99  \ getdrvparam-hd# (hd# -- fl) Parameter per Int13h/48h nach drvparam, fl=0: OK
100
101 \ q@ (ad -- n1 n2 n3 n4) Vierfachgenauen Wert als little-endian  aus ad im RAM
102 \ q! (n1 n2 n3 n4 ad --) Vierfachgenauen Wert als little-endian nach ad im RAM
103 \ q. (n1 n2 n3 n4 --)    Vierfachgenauen Wert als little-endian anzeigen
104
105 \ get#sects-hd#    (hd# -- q) Anzahl der Sektoren der Platte hd# aus drvparam
106 \ getbyte/sect-hd# (hd# -- n) Bytes pro Sektor    der Platte hd# aus drvparam
107 \ getsect/head-hd# (hd# -- d) Sektoren pro Kopf   der Platte hd# aus drvparam
```

```
\ gethead/cyli-hd# (hd# -- d) Koepfe pro Zylinder der Platte hd# aus drvparam
\ get#cylis-hd#    (hd# -- d) Anzahl der Zylinder der Platte hd# aus drvparam

\ (getLBAsect-hd#) (hd# dlba# -- fl) Maschinenprogrammkern zu getLBAsect-hd#
\ getLBAsect-hd#   (hd# dlba# -- fl) LBA-Sektor von hd# nach sectbuf, fl=0: OK
\ (putLBAsect-hd#) (hd# dlba# -- fl) Maschinenprogrammkern zu putLBAsect-hd#
\ putLBAsect-hd#   (hd# dlba# -- fl) LBA-Sektor von sectbuf nach hd#, fl=0: OK

\ getbyte/sect-fat16  (-- n) Anzahl Bytes pro Sektor, normalerweise 512d,
\ putbyte/sect-fat16  (n --) dito schreiben nach sectbuf
\ getsect/head-fat16  (-- n) Anzahl Sektoren pro Kopf, normalerweise 63d
\ putsect/head-fat16  (n --) dito schreiben nach sectbuf
\ getsect/clust-fat16 (-- c) Anzahl Sektoren pro Cluster aus sectbuf holen
\ putsect/clust-fat16 (c --) dito schreiben nach sectbuf
\ getsect/fat-fat16   (-- n) Anzahl Sektoren pro FAT aus sectbuf holen
\ putsect/fat-fat16   (n --) dito schreiben nach sectbuf
\ get#head-fat16      (-- n) Anzahl Koepfe aus sectbuf holen
\ put#head-fat16      (n --) dito schreiben nach sectbuf
\ get#fat-fat16       (-- c) Anzahl FATs aus sectbuf holen - immer c=2
\ put#fat-fat16       (c --) dito schreiben nach sectbuf
\ get#root-fat16      (-- n) Anzahl Datei-/Verzeichnis-Eintraege im Rootverz.
\ put#root-fat16      (n --) dito schreiben nach sectbuf
\ get#hidsect-fat16   (-- d) Anzahl Sektoren vor dem Bootsektor
\ put#hidsect-fat16   (d --) dito schreiben nach sectbuf
\ get#resvsect-fat16  (-- n) Anzahl reservierter Sektoren vor der ersten FAT
\ put#resvsect-fat16  (n --) dito schreiben nach sectbuf
\ getdrv#-fat16       (-- c) Festplatte (80) oder Diskette (00)?
\ putdrv#-fat16       (c --) dito schreiben nach sectbuf
\ getmedid-fat16      (-- c) Media-Descriptor. f8 = Festplatte, f0 = 3,5"-Disk
\ putmedid-fat16      (c --) dito schreiben nach sectbuf
\ get#small-fat16     (-- n) Anzahl Sektoren, die in 32 Bit Platz finden
\ put#small-fat16     (n --) dito schreiben nach sectbuf
\ get#large-fat16     (-- d) Anzahl Sektoren, die in 32 Bit keinen Platz finden
\ put#large-fat16     (d --) dito schreiben nach sectbuf

\ getbootprim-hd# (hd# part# -- fl) Bootsektor bei primaerer part#;    fl=0: OK
\ getEBR-hd#      (hd#  drv# -- fl) EBR des logischen  Laufwerks drv#; fl=0: OK
\ getbootext-hd#  (hd#  drv# -- fl) Bootsektor d. log. Laufwerks drv#; fl=0: OK


hex

variable  tmp1  \ zum (kurzzeitigen) Aufbewahren eines Parameters
variable  tmp2  \ dito
variable  hd#   \ Nummer der zuletzt eingegebenen Festplatte
variable  part# \ Nummer der zuletzt eingegebenen Partition
variable  drv#  \ Nummer des zuletzt eingegebenen logischen Laufwerks
2variable EBR   \ LBA-Nummer des zuletzt in sectbuf bearbeiteten EBRs
2variable EBR1  \ LBA-Nummer des Anfangs der erweiterten Partition
2variable boot  \ LBA-Nummer des zuletzt in sectbuf bearbeiteten Bootsektors
2variable LBA#  \ LBA-Nummer des zuletzt in sectbuf bearbeiteten Sektors

\ Zunaechst zwei Umwandlungsworte in allereinfachster Ausfuehrung, d.h. fuer
\ byte/sect = 200h . Ein- und Ausgabe auf dem Stack doppeltgenau:

: sect>kb ( dsects -- dkb ) d2/ ;  \ Sektoranzahl nach Kilobyte umwandeln
: kb>sect ( dkb -- dsects ) d2* ;  \ Kilobyte nach Sektoranzahl umwandeln

\ Das folgende Wort getmbr-hd# ist eine Erweiterung des Wortes getmbr aus
\ Teil 1 (VD-Heft 3/2008). getmbr bezog sich auf die Bootplatte, getmbr-hd#
```

```
\ laesst die freie Wahl der Festplatte (oder ZIP-Diskette oder USB-Stick) zu.
\ Der MBR liegt immer am Festplatten-Anfang. Es kann dafuer also bei der
\ CHS-Adressierung, mit den Werten C,H,S = 0,0,1 bleiben.

code (getmbr-hd#) ( hd# -- fl )       \ fl=0, wenn alles OK; sonst fl<>0
     ds push  es pop  dx pop  1 # cx mov  dh dh xor
     sectbuf # bx mov  201 # ax mov  13 int  ah al mov  ax push
     next end-code

: getmbr-hd# ( hd# -- fl )   \ fl=0, wenn alles OK; sonst fl<>0.
     depth 0 =               \ Stack leer, d.h.,
     if -1 exit then         \ Eingabe von hd# vergessen?
     dup hd# !               \ Eingabe hd# auch in die Variable hd#
     (getmbr-hd#) ;          \ Zum Interrupt 13h

\ Das folgende Wort putmbr-hd# ist genau das schreibende Gegenstueck zum
\ 'lesenden' Wort getmbr-hd# . Vorsicht: Erst ueberlegen, dann anwenden!

code (putmbr-hd#) ( hd# -- fl )       \ fl=0, wenn alles OK; sonst fl<>0
     ds push  es pop  dx pop  1 # cx mov  dh dh xor
     sectbuf # bx mov  301 # ax mov  13 int  ah al mov  ax push
     next end-code

: putmbr-hd# ( hd# -- fl )   \ fl=0, wenn alles OK; sonst fl<>0.
     depth 0 =               \ Stack leer, d.h.,
     if -1 exit then         \ Eingabe von hd# vergessen?
     dup hd# !               \ Eingabe hd# auch in die Variable hd#
     (putmbr-hd#) ;          \ Zum Interrupt 13h

\ Bei Veraenderungen am MBR ist man gut beraten, vorher die Worte mbr>bak
\ und bak>mbr aus Teil 1 anzuwenden, um den MBR notfalls aus der Datei
\ mbr.bak wieder hereinholen zu koennen. Es folgen Versionen fuer frei
\ waehlbare Festplatten hd#:

: mbr>bak-hd# ( hd# -- fl )             \ fl=0, wenn alles OK; sonst fl<>0
     getmbr-hd# " sectbuf dup 200 + save mbr.bak " $execute ;

: bak>mbr-hd# ( hd# -- fl )             \ fl=0, wenn alles OK; sonst fl<>0
     " open mbr.bak >r dsegment sectbuf 200 r> (get) close " $execute
     nip abort" Fehler!" putmbr-hd# ;

\ Das gleich folgende Wort getLBAsect# geht vom Inhalt des Sektorpuffers
\ sectbuf aus. (Welche Festplatte, ist dabei unwichtig.) Es wird angenommen,
\ dass in sectbuf ein gueltiger MBR liegt (keine Pruefung!). Es wird der
\ LBA-Anfangssektor der betrachteten Partition part# aus dem MBR in sectbuf
\ ausgelesen und auf den Datenstack gelegt. part# ist einfachgenau, dlba#
\ doppeltgenau. Das Wort kann auch zum Auslesen der Adresse eines EBRs (in
\ Sektoren relativ zum Anfang der erweiterten Partition) verwendet werden
\ (siehe getEBR-hd#).

\ Vorsicht: 12345678. xxxx 2! erscheint im RAM, also auch in sectbuf, an der
\ Adresse xxxx als 34-12-78-56 (kann mit dump ueberprueft werden). Also kein
\ reines 'Little Endian' (siehe auch Bemerkungen im Textteil des Artikels)!

\ Im folgenden Fehlerauffangwort testpart# wird ein unzulaessiges part#
\ (vorsichtshalber) durch part# = 1 ersetzt.

: testpart# ( part# -- part#/1 )     \ Wenn 0 < part# < 5, dann wird part#
     dup 1 5 within not              \ durchgereicht; wenn nicht, dann wird
     if cr ." Partition?" drop 1     \ die Meldung 'Partition?' ausgegeben und
```

```
228       then ;                          \ mit part#=1 gearbeitet.
229
230   \ Jetzt also das angekuendigte Forth-Wort getLBAsect# zum Einlesen des
231   \ relativen LBA-Anfangssektors der betrachteten Partition:
232
233   : getLBAsect# ( part# -- dlba# )      \ Anfangssektor der Partition part#.
234       testpart#                         \ Wenn nicht part# aus [1..4], dann
235       10 * sectbuf 1b6 + + 2@ swap      \ schaltet testpart# auf part#=1 um.
236       2dup lba# 2! ;                    \ dlba# nach lba# zur Reserve
237
238   \ Das folgende Forth-Wort getLBA#sect wirkt aehnlich wie das eben besprochene
239   \ getLBAsect#. Diesmal wird aber die Laenge (in Sektoren) der betreffenden
240   \ Partition eingeholt und auf den Datenstack und nach lba# gelegt.
241
242   : getLBA#sect ( part# -- d#lba )      \ Sektoranzahl in der Partition part#
243       testpart#                         \ Wenn nicht part# aus [1..4], dann
244       10 * sectbuf 1ba + + 2@ swap      \ schaltet testpart# auf part#=1 um.
245       2dup lba# 2! ;                    \ d#lba nach lba# zur Reserve
246
247   \ Bei der Bildschirmanzeige der Stackwerte reicht hier i.A. das Wort d.
248   \ nicht. Es ist ud. zu verwenden. Der einfache Punkt waere ganz falsch.
249
250   \ Erhaelt man bei getLBAsect# den Wert 00000000, dann hat man es mit einem der
251   \ folgenden drei Faelle zu tun:
252
253   \ (1) Regelfall und betreffende Partition part# nicht vorhanden,
254   \ (2) 'Logisches' Laufwerk der erweiterten Partition und part# = 3 oder 4,
255   \ (3) Fehler in der Partitionstabelle MBR oder EBR.
256
257   \ Entsprechendes gilt fuer getLBA#sect .
258
259   \ Die folgenden beiden Forth-Worte sind die Gegenstuecke zu getLBAsect# und
260   \ getLBA#sect . Ausgehend von der Partitionsnummer part# auf dem Datenstack
261   \ werden diese Werte an die dafuer vorgesehene Stelle der Partitionstabelle
262   \ gelegt.
263
264   \ Die Operationen wirken zunaechst nur auf den Sektorpuffer sectbuf. Gefahr
265   \ bestuende erst dann, wenn man (zum Abschluss) eines der 'direkten' put-Worte,
266   \ wie putmbr oder putmbr-hd#, ins Spiel bringt.
267
268   : putLBAsect# ( dlba# part# -- )      \ Anfangssektor der Partition part#.
269       testpart#                         \ Umschalten auf part#=1, wenn nicht
270       10 * sectbuf 1b6 + + rot rot      \ part# aus [1..4].
271       2dup lba# 2! rot rot swap 2! ;    \ dlba# in 2Variable lba# in Reserve
272
273   : putLBA#sect ( d#lba part# -- )      \ Sektoranzahl in der Partition part#.
274       testpart#                         \ Umschalten auf part#=1, wenn nicht
275       10 * sectbuf 1ba + + rot rot      \ part# aus [1..4].
276       2dup lba# 2! rot rot swap 2! ;    \ d#lba in 2Variable lba# in Reserve
277
278   \ Die folgenden Forth-Worte erwarten den MBR im Puffer sectbuf. Die Nummer hd#
279   \ der Platte wird dann nicht mehr benoetigt. Es muss also vorher getmbr-hd#
280   \ und hinterher putmbr-hd# in Aktion treten. Wichtig ist die einzugebende
281   \ Partitions-Nummer part#. Ich habe ueberall, wo sinnvoll, das Forth-Wort
282   \ testpart# eingebaut. Damit wird ueberprueft, ob part# wirklich zulaessig ist.
283   \ Wenn nicht, erscheint die Warnung 'Partition?' und es wird part#=1 gesetzt.
284
285   \ Von MS-DOS aus hat man den Zugriff nur auf Partitionen des Typs 01, 04, 05
286   \ und 06 (oder 16 usw., falls 'versteckt'). Es bedeutet dabei:
287
```

```
288  \ 01 = FAT12, Primaerpart. oder log. Laufwerk mit weniger als 32680 Sektoren,
289  \ 04 = FAT16, dito mit 32680 bis 65535 Sektoren (16-33 MB),
290  \ 05 = erweiterte Partition,
291  \ 06 = FAT16, mehr als 65535 Sektoren (33 MB bis 4 GB),
292  \ 07 = NTFS, primaer oder erweitert,
293  \ 0C = FAT32, mit erweitertem Interrupt 13h,
294  \ 0E = FAT16, BIGDOS mit erweitertem Interrupt 13h,
295  \ 0F = erweiterte Partition mit erweitertem Interrupt 13h.
296
297  : gettyp    ( part# -- n )   \ Holt den Dateisystem-Typus  aus part# in sectbuf
298      testpart# dup part# !   \ part# aus [1..4] ? Aufbewahren in Variable part#
299      10 * sectbuf 1b2 + + c@ ; \ Bei 16 ('versteckt' 06) wird 1 mitgefuehrt usw
300
301  \ Bei dem folgenden Forth-Wort puttyp (das lediglich auf den Puffer sectbuf
302  \ wirkt, nicht auf die Festplatte) kann man ohne Weiteres auch andere Typ-Werte
303  \ als die oben angegebenen 'DOS-Werte' verwenden. Das wird beispielsweise fuer
304  \ Linux interessant: Linux native = 83, Linux swap = 82.
305
306  : puttyp    ( n part# -- )   \ Legt den Dateisystem-Typus nach part# in sectbuf
307      testpart# dup part# !   \ part# aus [1..4] ? Aufbewahren in Variable part#
308      10 * sectbuf 1b2 + + c! ; \ Bei 16 muss die 1 mit angegeben werden usw
309
310  \ Und nun ein paar spezielle Faelle zum Einbringen oder Aendern von Typnummern
311  \ in (vorerst nur) sectbuf. Eine eventuelle 1 fuer 'versteckt' bleibt erhalten.
312
313  : putfat16  ( part# -- )     \ Schreibt 6 in das untere Halbbyte an Offset 4
314      testpart# dup part# !   \ part# aus [1..4] ?
315      >r r@ gettyp f0 and 06 or r> puttyp ;
316  : putfat32  ( part# -- )     \ Schreibt c in das untere Halbbyte an Offset 4
317      testpart# dup part# !   \ part# aus [1..4] ?
318      >r r@ gettyp f0 and 0c or r> puttyp ;
319  : putntfs   ( part# -- )     \ Schreibt 7 in das untere Halbbyte an Offset 4
320      testpart# dup part# !   \ part# aus [1..4] ?
321      >r r@ gettyp f0 and 07 or r> puttyp ;
322
323  \ Die beiden folgenden Forth-Worte beziehen sich auf Linux. 'Versteckt' gilt
324  \ hier nicht.
325
326  : putlin83  ( part# -- )     \ Schreibt 83 in das Byte mit Offset 4 in part#
327      testpart# dup part# !   \ part# aus [1..4] ?
328      83 swap puttyp ;
329  : putlin82  ( part# -- )     \ Schreibt 82 in das Byte mit Offset 4 in part#
330      testpart# dup part# !   \ part# aus [1..4] ?
331      82 swap puttyp ;
332
333  \ Bootsektoren, MBR und EBRs (extended boot records) werden mit 55 aa (der
334  \ 'Magic Number') in Offset 1fe und 1ff abgeschlossen.
335
336  : put55-aa  ( -- )           \ Magic Number nach sectbuf
337      aa55 sectbuf 1fe + ! ;
338
339  : testfat16 ( part# -- fl )  \ Prueft (fl=0, wenn nicht), ob im unteren
340      testpart# dup part# !   \ Halbbyte von part#/Offset 4 der Wert 6 liegt.
341      sectbuf 1b2 + c@ 0f
342                    and 06 = ;
343  : testfat32 ( part# -- fl )  \ Prueft (fl=0, wenn nicht), ob im unteren
344      testpart# dup part# !   \ Halbbyte von part#/Offset 4 der Wert c liegt.
345      sectbuf 1b2 + c@ 0f
346                    and 0c = ;
347  : testntfs  ( part# -- fl )  \ Prueft (fl=0, wenn nicht), ob im unteren
```

```
348       testpart# dup part# !   \ Halbbyte von part#/Offset 4 der Wert 7 liegt.
349       sectbuf 1b2 + c@ 0f
350                     and 07 = ;
351   : testlin83 ( part# -- fl )  \ Prueft (fl=0, wenn nicht), ob im Byte von
352       testpart# dup part# !   \ part#/Offset 4 der Wert 83 liegt.
353       sectbuf 1b2 + c@ 83 = ;
354   : testlin82 ( part# -- fl )  \ Prueft (fl=0, wenn nicht), ob im Byte von
355       testpart# dup part# !   \ part#/Offset 4 der Wert 82 liegt.
356       sectbuf 1b2 + c@ 82 = ;
357   : test55-aa ( -- fl )        \ Prueft (fl=0, wenn nicht), ob ab Offset 1fe
358       sectbuf 1fe + @ aa55 = ; \ von sectbuf die beiden Bytes 55 aa liegen.
359
360   \ Das Forth-Wort getdrvparam-hd# weiter unten legt die ermittelten Parameter im
361   \ gleich besprochenen Puffer drvparam ab. Nach erfolgter Operation (bis zum
362   \ naechsten Aufruf von getdrvparam-hd#) koennen sie von dort aus auch beliebig
363   \ oft wieder abgeholt werden. Dem Puffer drvparam habe ich mit Absicht die
364   \ Laenge eines 'ueblichen' Sektors (512d Bytes) gegeben, damit ich drvparam
365   \ notfalls (neben sectbuf) auch als zusaetzlichen Sektorpuffer einsetzen kann.
366   \ Die Konstruktion der Forth-Konstanten drvparam gleicht hier der von sectbuf.
367
368   210 allot here    \ Platz fuer mind. 1 Sektor = 512d Bytes
369   here 0f and -     \ drvparam an Paragraphenanfang
370   200 -             \ Anfang des Sektorpuffers
371   constant drvparam \ Liefert Adresse des Parameterpuffers
372
373   \ In Analogie zu showsectbuf aus Teil 1 noch schnell das Wort showdrvparam :
374
375   : showdrvparam    ( -- ) drvparam 200 dump ;
376   : showdrvparam100 ( -- ) drvparam 100 dump ;
377
378   \ Das jetzt folgende Forth-Wort getdrvparam-hd# und die damit verbundenen Worte
379   \ beziehen ihre Werte ueber den Interrupt 13h/48h und koennen zur Ueberpruefung
380   \ der entsprechenden Eintragungen in den Datentabellen der Bootsektoren
381   \ verwendet werden (vergleiche weiter unten).
382
383   \ Die Anwendung von getdrvparam-hd# ist nicht nur auf Bootplatten beschraenkt.
384   \ Ich habe das mit einem von der Platte hd#=80 gebooteten Windows-ME-System und
385   \ einem ueber hex 81 getdrvparam-hd# abgefragten festplatten-formatierten
386   \ USB-Stick ueberprueft. (Win-ME gehoert noch zu denjenigen Windows-Systemen,
387   \ die sich, anders als XP, bei direktem Festplatten-Zugriff nicht beschweren.)
388
389   code (getdrvparam-hd#) ( hd# -- fl )      \ fl=0, wenn alles OK
390                 dx pop   \ Festplattennummer hd# (80...ff) nach dl
391              dh dh xor   \ Vorsichtshalber dh=0
392                 si push  \ si (=ip) sicherstellen
393      drvparam # si mov   \ ds:si mit ds:drvparam laden
394          4800 # ax mov   \ Funktion 'get drive parameters' (LBA)
395                 13 int   \ Interrupt aufrufen
396                 si pop   \ si (=ip) wiederherstellen
397              ah al mov   \ Jetzt ax = 0 bei Erfolg
398                 ax push  \ ax <> 0  => Fehler
399           next end-code
400
401   : getdrvparam-hd# ( hd# -- fl ) \ fl=0, wenn alles OK; sonst fl<>0.
402       depth 0 = if -1 then       \ Eingabe von hd# vergessen und Stack leer?
403       dup hd# !                  \ Eingabe hd# auch in die Variable hd#
404       (getdrvparam-hd#) ;        \ Zum Interrupt 13h/48h
405
406   \ Aus Offset 10h der Tabelle drvparam erhaelt man die Gesamtzahl der Sektoren
407   \ der Festplatte hd#. Dafuer sind 8 Bytes vorgesehen. Das entspricht einem
```

```
\ vierfachgenauen Wert (64 Bit). Wegen Fragen bei der Darstellung von
\ vierfachgenauen Werten (Little-Endian?) vergleiche den  Textteil des
\ vorliegenden Artikels.

\ Das folgende Forth-Wort q@ holt vom RAM-Speicher ab Adresse ad den dort in
\ Little-Endian-Darstellung als Hex-Byte-Folge zu interpretierenden 64-Bit-Wert
\ auf den Datenstack, und zwar so, dass er im Stack-RAM-Bereich (ich gehe von
\ einem nach kleineren Adressen hin wachsenden Stack aus) wieder als
\ Little-Endian-Bytefolge auftritt.

\ Mit sp@ wollte ich hier und anderswo in diesem Artikel nicht arbeiten, da
\ mir dieses Wort als zu systemabhaengig und daher inkompatibilitaetstraechtig
\ erschien (in den ANS-Forth-Empfehlungen wird es nicht erwaehnt).

: q@ ( ad -- n1 n2 n3 n4 )
    8 + tmp1 ! 4 0 do tmp1 @ 2 - dup tmp1 ! @ loop ;

\ Das jetzt folgende Wort q! ist die Umkehrung zu q@. Die Eintragungen auf dem
\ Datenstack seien so wie bei q@ zu interpretieren. Fuehrende Nullen brauchen
\ in keinem dieser Viererpakete eingegeben zu werden. In der (ueber q!
\ erzeugten) RAM-Darstellung und dann auch (bei Anwendung von q.) auf dem
\ Bildschirm erscheint aber jede einzelne Null, auch eventuell vorhandene
\ fuehrende Nullen der Gesamtanzeige. Der Einfachheit halber seien nur Zahlen
\ ohne Vorzeichen zugelassen. Und die Viererpakete erhalten nur dann ihren
\ eigentlichen Sinn, wenn sie als Hexzahlen auftreten.

: q! ( n1 n2 n3 n4 ad -- )
    2 - tmp1 ! 4 0 do tmp1 @ 2 + dup tmp1 ! ! loop ;

\ Das folgende Forth-Wort q. liefert eine schreibgerechte Anzeige von
\ vierfachgenauen Werten auf dem Bildschirm nach der besprochenen Methode.

: q. ( n1 n2 n3 n4 -- )
    base @ tmp1 !  hex
    swap 2swap swap <# 8 0 do # loop #> type
                    <# 8 0 do # loop #> type space
    tmp1 @ base ! ;

\ Und nun wieder zu den Festplatten-Parametern. d ist auf dem Stack wie ueblich
\ als d = n1n2 = n2 n1 zu lesen, q aber als q = n1n2n3n4 = n1 n2 n3 n4 . Im
\ Fehlerfall ist q = ffff ffff ffff ffff und es wird 'HD-Fehler' angezeigt.

: get#sects-hd# ( hd# -- q )           \ Anzahl der Sektoren der Platte hd#
    getdrvparam-hd#
    if                                 \ HD-Fehler? Ja, dann Ausstieg
      cr ." HD-Fehler!" -1. 2dup exit  \ mit q = ffff ffff ffff ffff
    then
    drvparam 10 + q@ ;

\ Eine Ueberpruefung meiner Festplatte per 80 get#sects q. ergab den Hexwert
\ 2629ffb als Gesamtzahl aller Sektoren. Sektorgroesse ist 512 Bytes dezimal.
\ Auf das Kilobyte gehen also 2 Sektoren. Das liefert 1314ffd Kilobyte als
\ Hexwert und somit 20008957 als dezimale Anzahl aller Kilobytes. Das sind rund
\ 20 Gigabyte, was genau der Kapazitaet der von mir fuer diese Experimente
\ eingesetzten Festplatte entspricht.

\ Und nun der Rest der vom Interrupt 13h/48h gelieferten HD-Parameter:

: getbyte/sect-hd# ( hd# -- n )  \ Zahl der Bytes pro Sektor auf hd#
    getdrvparam-hd# abort" Fehler!" drvparam 18 +  @      ;
```

```
: getsect/head-hd# ( hd# -- d )  \ Zahl der Sektoren pro Kopf auf hd#
    getdrvparam-hd# abort" Fehler!" drvparam 0c + 2@ swap ;
: gethead/cyli-hd# ( hd# -- d )  \ Zahl der Koepfe pro Zylinder auf hd#
    getdrvparam-hd# abort" Fehler!" drvparam 08 + 2@ swap ;
: get#cylis-hd#    ( hd# -- d )  \ Gesamtzahl der Zylinder auf hd#
    getdrvparam-hd# abort" Fehler!" drvparam 04 + 2@ swap ;

\ Wie schon erwaehnt, werdem vom System in den Datentabellen der Bootsektoren
\ ebenfalls noch einige Parameter der Festplatte festgehalten. Nur an wenige
\ davon kommt man auch mit dem Interrupt 13h/48h heran. Jetzt noch schnell
\ ein paar Worte ueber eben solche Parameter aus den Bootsektoren.

\ Gerade bei den Festplatten-Parametern waere es interessant, nicht nur die
\ bootende Festplatte, sondern auch eine eventuell eingesetzte zweite
\ Festplatte (z.B. einen im System eingebundenen USB-Stick oder ein als
\ Festplatte formatiertes Zip-Laufwerk) zu beruecksichtigen. Dazu brauche ich
\ ein lesendes und ein schreibendes Forth-Wort fuer beliebige LBA-Sektoren, im
\ Gegensatz zu Teil 3 diesmal aber bei vorgebbarer Festplatten-Nummer hd#. Ich
\ hole das jetzt nach. Die \-Kommentare laufen wie in Teil 3 (VD-Heft 1/2009)
\ bei getLBAsect und putLBAsect und koennen hier weggelassen werden.

\ Das folgende Forth-Wort getLBAsect-hd# holt von der Festplatte hd# (80,81,..,
\ einfachgenau) den Sektor mit der Nummer dlba# (0,1,2,.., doppeltgenau, d.h.
\ Punktzahl!) und legt ihn in den Puffer sectbuf. hd# wird (bis auf das
\ naechste Forth-Wort mit dem Eingabe-Parameter hd#) in der gleichnamigen
\ Variablen hd# aufbewahrt. dlba# wird (bis zur naechsten dlba#-Verarbeitung)
\ in der 2Variablen lba# gespeichert.

\ Achtung: Es werden auch in diesem Artikel jeweils nur immer einzelne Sektoren
\ beruecksichtigt. Mehrere Sektoren in einem einzigen Durchgang kann man sich
\ leicht dadurch beschaffen, dass man in der Befehlsfolge 1 # dap 02 + #) mov
\ im folgenden Wort (getLBAsect-hd#) den Wert 1 entsprechend erhoeht und
\ dafuer sorgt, dass der Puffer sectbuf entsprechend groesser angesetzt wird.

code (getLBAsect-hd#) ( hd# dlba# -- fl )  \ fl=0, wenn alles OK
      18       # dap      #) mov
      1        # dap 02 + #) mov
      sectbuf  # dap 04 + #) mov
      csegment # dap 06 + #) mov
      ax pop  ax dap 0a + #) mov  ax lba#      #) mov
      ax pop  ax dap 08 + #) mov  ax lba# 02 + #) mov
      0        # dap 0c + #) mov
      0        # dap 0e + #) mov
      dx pop
      dx hd#  #) mov  dh dh xor  si push  dap # si mov  4200 # ax mov
      13 int  si pop  ah al mov  ax push  next end-code

: getLBAsect-hd# ( hd# dlba# -- fl )  \ fl=0, wenn alles OK
    depth 3 <                         \ hd# oder/und dlba# vergessen?
    if                                \ Ja, dann Stack saeubern und
      begin depth 0 >
        if drop 0 else -1 then
      until
      cr ." Fehler!" -1 exit          \ Ausstieg unter fl=ffff
    then
    (getLBAsect-hd#) ;

\ getLBAsect aus Teil 3 bezieht sich auf die Bootplatte, getLBAsect-hd# hier
\ auf die beliebig vorgebbare Festplatte mit der Nummer hd#. dap (disk address
\ packet) ist der Parameter-Uebergabe-Puffer aus Teil 3 (VD-Heft 1/2009).
```

```
528
529  \ Achtung: dlba# muss doppeltgenau (als 'Punktzahl') eingegeben werden!
530
531  \ In Offset 'dap 04 +' von dap wird die 'Lang-'Adresse des Uebertragungspuffers
532  \ in der Form segment:sectbuf (bei mir beispielsweise 2860:6750) festgehalten,
533  \ und zwar in der Little-Endian-Form 50 67 60 28 . Steht hier der Wert
534  \ ffff:ffff, dann soll laut 'Browns Liste' ab Offset 'dap 10 +' die lineare
535  \ 64-Bit-Adresse des Uebertragungspuffers abgelegt werden.
536
537  \ Und hier als Schreib-Pendant zu (getLBAsect-hd#) das Wort (putLBAsect-hd#).
538  \ (Vorsicht vor unueberlegtem Tastendruck!) Der einzige Unterschied zu
539  \ (getLBAsect-hd#) liegt in 4300 statt 4200. Vielleicht koennte (sollte) man
540  \ putLBAsect-hd# und (putLBAsect-hd#) durch 'einfaches' Patchen aus
541  \ getLBAsect-hd# und (getLBAsect-hd#) heraus erzeugen?
542
543  code (putLBAsect-hd#) ( hd# dlba# -- fl )  \ fl=0, wenn alles OK
544        18 #       dap #)      mov
545        1        # dap 02 + #) mov
546        sectbuf  # dap 04 + #) mov
547        csegment # dap 06 + #) mov
548        ax pop  ax dap 0a + #) mov  ax lba#     #) mov
549        ax pop  ax dap 08 + #) mov  ax lba# 2 + #) mov
550        0        # dap 0c + #) mov
551        0        # dap 0e + #) mov
552        dx pop
553        dx hd#  #) mov  dh dh xor  si push  dap # si mov  4300 # ax mov
554        13 int  si pop  ah al mov  ax push  next end-code
555
556  : putLBAsect-hd# ( hd# dlba# -- fl )  \ fl=0, wenn alles OK
557      depth 3 <                         \ hd# oder/und dlba# vergessen?
558      if                                \ Ja, dann Stack saeubern und
559        begin depth 0 >
560          if drop 0 else -1 then
561        until
562        cr ." Fehler!" -1 exit          \ Ausstieg unter fl=ffff
563      then
564      (putLBAsect-hd#) ;
565
566  \ Die jetzt folgenden Forth-Worte bearbeiten Parameter aus den Bootsektoren.
567  \ Ich gehe davon aus, dass im Sektorpuffer sectbuf ein (primaerer oder
568  \ erweiterter) Bootsektor liegt. Festplatten- oder die Partitions-Nummern
569  \ werden also nicht benoetigt: Die Namen der betreffenden Forth-Worte haben
570  \ keine Endung -hd#. Das Einholen von Bootsektoren in den Puffer sectbuf (unter
571  \ Einbeziehung von hd# und part#) wird am Schluss des Artikels besprochen.
572
573  \ Jeder Bootsektor enthaelt einen BIOS-Parameter-Block, den BPB. Ich bin hier
574  \ nur an FAT16 interessiert. FAT32 kommt spaeter dran. Die Sektoren werden als
575  \ LBA-Sektoren (Zaehlung 0,1,2,...) aufgefasst.
576
577  : getbyte/sect-fat16  ( -- n )  \ Anzahl Bytes pro Sektor, normalerweise 512d,
578      sectbuf 0b +  @ ;           \ aber auch 1024, 2048 oder 4096 sind zulaessig.
579  : putbyte/sect-fat16  ( n -- )  \ Anzahl Bytes pro Sektor, ... dito
580      sectbuf 0b +  ! ;
581  : getsect/head-fat16  ( -- n )  \ Anzahl Sektoren pro Kopf, normalerweise 63d
582      sectbuf 18 +  @ ;
583  : putsect/head-fat16  ( n -- )  \ Anzahl Sektoren pro Kopf, ... dito
584      sectbuf 18 +  ! ;
585  : getsect/clust-fat16 ( -- c )  \ Anzahl Sektoren pro Cluster. Nur bis zu 65536
586      sectbuf 0d + c@ ;           \ Cluster sind moeglich. Gueltige Werte sind
587                                  \ 1,2,4,8,16,32,64,128 (alles dezimal).
```

```
588  : putsect/clust-fat16 ( c -- )  \ Anzahl Sektoren pro Cluster, ... dito
589      sectbuf 0d + c! ;
590  : getsect/fat-fat16   ( -- n )  \ Anzahl Sektoren pro FAT aus sectbuf holen
591      sectbuf 16 +  @ ;
592  : putsect/fat-fat16   ( n -- )  \ Anzahl Sektoren pro FAT  in sectbuf legen
593      sectbuf 16 +  ! ;
594  : get#head-fat16      ( -- n )  \ Anzahl Koepfe aus sectbuf holen
595      sectbuf 1a +  @ ;
596  : put#head-fat16      ( n -- )  \ Anzahl Koepfe  in sectbuf legen
597      sectbuf 1a +  ! ;
598  : get#fat-fat16       ( -- c )  \ Anzahl FATs aus sectbuf holen
599      sectbuf 10 + c@ ;           \ In FAT16 immer c=2
600  : put#fat-fat16       ( c -- )  \ Anzahl FATs  in sectbuf legen
601      sectbuf 10 + c! ;           \ In FAT16 immer c=2
602  : get#root-fat16      ( -- n )  \ Anzahl der Datei- oder Verzeichnis-Eintraege,
603      sectbuf 11 +  @ ;           \ die im Root-Verzeichnis gespeichert werden
604                                  \ koennen (je 32 Bit), normalerweise bis zu
605                                  \ 512d. Lange Dateinamen (ab Windows 95)
606                                  \ benoetigen mehrere 32-Bit-Eintraege.
607  : put#root-fat16      ( n -- )  \ ... dito
608      sectbuf 11 +  ! ;
609
610  : get#hidsect-fat16   ( -- d )  \ Anzahl Sektoren vor dem Bootsektor. Daraus
611      sectbuf 1c + 2@ swap ;      \ abs. Offset zum Root-Verzeichnis ermittelbar.
612  : put#hidsect-fat16   ( d -- )  \ ... dito
613      swap sectbuf 1c + 2! ;
614  : get#resvsect-fat16  ( -- n )  \ Reservierte Sektoren: Zahl der Sektoren, mit
615      sectbuf 0e +  @ ;           \ Bootsektor, die der ersten FAT vorausgehen.
616  : put#resvsect-fat16  ( n -- )  \ Reservierte Sektoren: ... dito
617      sectbuf 0e +  ! ;
618  : getdrv#-fat16       ( -- c )  \ c aus sectbuf holen. Zur Unterscheidung von
619      sectbuf 24 + c@ ;           \ Festplatte (80) oder Diskette (00). Nur fuer
620                                  \ Bootlaufwerke von Bedeutung.
621  : putdrv#-fat16       ( c -- )  \ c nach sectbuf legen. ... dito
622      sectbuf 24 + c! ;
623  : getmedid-fat16      ( -- c )  \ Media-Descriptor. f8 = Festplatte,
624      sectbuf 15 + c@ ;           \ f0 = High-Density-3,5"-Diskette,
625  : putmedid-fat16      ( c -- )  \ ... dito: c nach sectbuf legen.
626      sectbuf 15 + c! ;
627  : get#small-fat16     ( -- n )  \ Anzahl Sektoren der Partition, die in 32 Bit
628      sectbuf 13 +  @ ;           \ Platz finden (n<65536). Partitionen mit mehr
629                                  \ als 65536 Sektoren haben hier den Wert 0 und
630                                  \ get#large-fat16 wird wirksam.
631  : put#small-fat16     ( n -- )  \ ... dito: n nach sectbuf legen.
632      sectbuf 13 +  ! ;
633  : get#large-fat16     ( -- d )  \ Anzahl Sektoren der Partition, die in 32 Bit
634      sectbuf 20 + 2@ swap ;      \ keinen Platz finden (n>65536). Partitionen
635                                  \ mit weniger als 65536 Sektoren haben hier den
636                                  \ Wert 0 und get#small-fat16 wird wirksam.
637  : put#large-fat16     ( d -- )  \ ... dito: d nach sectbuf legen.
638      swap sectbuf 20 + 2! ;
639
640  \ Einige dieser Parameter aus dem Bootsektor koennen (siehe oben) auch ueber
641  \ den Interrupt 13h/48h ermittelt werden. Dieser erscheint mir fuer einen
642  \ Reparaturfall als vertrauenswuerdiger, da er nicht so leicht von
643  \ zerschossenen Tabellen auf der Festplatte beeinflusst werden kann. Oder
644  \ irre ich mich? Ueber Hinweise wuerde ich mich freuen.
645
646  \ Das folgende Forth-Wort getbootprim-hd# laedt den Bootsektor der primaeren
647  \ Partition part# der Platte hd# in den Puffer sectbuf. Die genaue Funktion,
```

```
\ auch die eingebaute Fehler-Erkennung und -Vermeidung geht aus den
\ Kommentaren  hervor. Die Sektor-Adresse des Bootsektors wird in der
\ 2Variablen boot aufbewahrt, die Plattennummer hd# in der Variablen hd#.

: getbootprim-hd# ( hd# part# -- fl )    \ fl=0, wenn alles OK
    depth 0 =                            \ hd# und part# vergessen?
    if -1 exit then                      \ Ja, dann mit fl=ffff raus.
    part# !                              \ part# auch in die Variable part#
    depth 0 =                            \ hd# oder part# vergessen?
    if -1 exit then                      \ Ja, dann mit fl=ffff raus.
    hd# !                                \ Eingabe hd# auch in die Variable hd#
    part# @ testpart# part# !            \ 0 < part# < 5 ? Nein, dann part#=1
    hd# @ getmbr-hd#                     \ Versuch, MBR zu holen
    if cr ." HD-Fehler!" -1 exit then    \ HD-Fehler? Ja, dann Ausstieg unter
                                         \ fl=ffff. Andernfalls liegt jetzt der
                                         \ MBR im Sektorpuffer sectbuf
    sectbuf 1ae + part# @                \ Ausgangsposition im Puffer sectbuf
    10 * + 4 + c@ dup                    \ Offset 04 in MBR-Partitionszeile
    0 = swap 0f and 5 = or               \ Typ 05 oder 00 ?
    if cr ." Keine primaere Partition!" \ Ja, dann Fehler
       -1 exit                           \ und Ausstieg unter fl=ffff
    then
    hd# @ part# @ getLBAsect#            \ Ansonsten jetzt Adr(Sektor-Nr.)
    2dup boot 2!                         \ in der 2Variablen boot aufbewahren
    getLBAsect-hd#                       \ und dann den Bootsektor der
    dup if cr ." HD-Fehler!" then ;      \ Partition part# nach sectbuf holen

\ Und jetzt noch die Bootsektoren der erweiterten Partition (vergleiche das
\ im Textteil ueber die Festplatten-Organisation Gesagte. Das Thema wurde
\ schon bei getboot aus dem VD-Heft 3/2008 behandelt. Hier fuehre ich frei
\ waehlbare Platten (nicht nur hd#=80), Fehler-Flags im Zusammenhang mit
\ beweglichen Teilen und LBA-Sektoren (statt CHS) ein.

\ Wichtig beim gleich folgenden Forth-Wort getEBR-hd#, das den EBR (extended
\ boot record) des logischen Laufwerks drv# der erweiterten Partition von hd#
\ in den Puffer sectbuf legt, sind vier Werte in den ersten beiden Zeilen der
\ 'Partitionstabelle' (Offset 1be bis einschliesslich 1dd). Diese Tabelle
\ des EBRs stimmt mit der Aufteilung der Tabelle des MBRs ueberein. Die Zeilen
\ 3 und 4 bleiben aber beim EBR auf 0 gesetzt. Auch die anderen Bytes, bis auf
\ die 'Magic Number' 55 aa bei Offset 1fe und 1ff, enthalten lauter Nullen.

\ Offset 1c6: Abstand in Sektoren vom EBR (Laufwerksanfang) zum Bootsektor,
\ Offset 1ca: Laenge in Sektoren des logischen Laufwerks,
\ Offset 1d6: Abstand in Sektoren vom Anfang der erweiterten Partition zum
\             Anfang des naechsten logischen Laufwerks (zu dessem EBR),
\ Offset 1da: Laenge in Sektoren der gesamten erweiterten Partition.

\ Ist die ganze Zeile des EBRs, in welcher die Eintragungen bei Offset 1d6 und
\ 1da zu finden sein koennen, mit 0 belegt, dann war das betreffende logische
\ Laufwerk das letzte in der erweiterten Partition.

\ EBRs kann es nur in der erweiterten Partition geben. Die Bezeichnung des
\ folgenden Wortes getEBR-hd# kann also entsprechend kurz gehalten werden. Hat
\ man den EBR des gewuenschten logischen Laufwerks, dann kann man aus Offset
\ 1c6 und der LBA-Nummer des EBRs die LBA-Nummer des zugehoerigen Bootsektors
\ gewinnen (siehe gleich). Primaere Partitionen beginnen bei ihrem Bootsektor:
\ Man kann sagen, ihre EBRs sind in den vier Zeilen der Partitionstabelle des
\ MBRs enthalten.

\ Beim folgenden getEBR-hd# wird zunaechst versucht, den MBR der Platte hd#
```

```
\ nach sectbuf zu speichern (bei Fehler Ausstieg mit fl=ffff und der Meldung
\ 'HD-Fehler!'). Gelingt es, dann wird die Partitionsnummer der erweiterten
\ Partition ermittelt (bei Fehler Ausstieg mit fl=ffff und einer Meldung). Dann
\ geht es an die Ermittlung der LBA-Adresse des EBRs des Laufwerks drv#.

\ Dazu muss die verkettete Liste der EBRs bis zum logischen Laufwerk drv#
\ durchlaufen werden und dann ist im EBR zu drv# noch die EBR-Adresse von drv#
\ (in Sektoren ab dem Anfang der erweiterten Partition) zu ermitteln. Sodann
\ wird der EBR zu drv# in den Sektorpuffer sectbuf geholt.

\ Wurde bei der Stack-Eingabe fuer drv# (irrtuemlich) eine Zahl gewaehlt, die
\ groesser ist als die vorhandene Zahl von logischen Laufwerken, dann behaelt
\ sectbuf den EBR des letzten gerade noch vorhandenen logischen Laufwerks, es
\ wird die Fehlermeldung 'Letztes logisches Laufwerk = drv#' (mit der dann
\ geltenden konkreten Zahl fuer drv#) ausgegeben und getEBR-hd# wird mit
\ fl=ffff verlassen. Klappt alles, dann bleibt fl=0 auf dem Stack.

\ In jedem Fall wird die letzte noch brauchbare EBR-Adresse (als Sektor-Nummer)
\ in der 2Variablen EBR festgehalten, von wo aus sie beispielsweise per hd#
\ drv# getEBR-hd# (siehe folgendes Forth-Wort) in Verbindung mit einem
\ anschliessenden 1 getlbasect# bei der Ermittlung des zugehoerigen Bootsektors
\ helfen kann.

\ Hier und anderswo haette man -rot erwartet. In ANS-Forth gibt es kein -rot.
\ Ich schreibe dafuer rot rot und bin ANS-kompatibel.

: getEBR-hd# ( hd# drv# -- fl )          \ fl=0, wenn alles OK
    depth 0 =                            \ hd# und drv# vergessen?
    if -1 exit then                      \ Ja, dann mit fl=ffff raus.
    drv# !                               \ drv# auch in die Variable drv#
    depth 0 =                            \ hd# oder drv# vergessen?
    if -1 exit then                      \ Ja, dann mit fl=ffff raus.
    hd# !                                \ Eingabe hd# auch in die Variable hd#
    hd# @ getmbr-hd#                     \ Versuch, MBR zu holen
    if cr ." HD-Fehler!" -1 exit then    \ HD-Fehler? Ja, dann Ausstieg unter
                                         \ fl=ffff. Andernfalls liegt jetzt der
                                         \ MBR im Sektorpuffer sectbuf
    0 tmp1 !                             \ tmp1 ist jetzt Partitionszaehler
    begin
      sectbuf 1ae + tmp1 @ 1 + tmp1 !
      tmp1 @ 4 >                         \ Schon mehr als 4 MBR-Zeilen?
      if                                 \ Ja, dann Fehlermeldung, Zaehlwert
        cr ." Erweiterte Partition nicht vorhanden!"
        drop -1 exit                     \ entfernen und Ausstieg mit fl=ffff;
      then                               \ ansonsten
      tmp1 @ 10 * + 4 + c@ 0f and 05 =   \ Offset 04 in MBR-Partitionszeile
    until                                \ = 05, dann raus: tmp1 = # erw. Part.
    tmp1 @ getLBAsect#                   \ Partitionsanfang der erw. Partition
    EBR1 2!                              \ LBA-Nummer in die 2Variable EBR1
    hd# @ EBR1 2@ getLBAsect-hd#         \ Den ersten EBR nach sectbuf holen
    if cr ." HD-Fehler!" -1 exit         \ HD-Fehler? Dann Meldung und
    then                                 \ Ausstieg mit fl=ffff
    1 tmp1 !                             \ Nun ist tmp1 Zaehler fuer die EBRs
    begin                                \ Verkettete EBR-Liste
      hd# @ EBR1 2@ tmp1 @               \ Wenn tmp1 bereits > 1, dann naechste
      1 <> if 2 getLBAsect# d+ then      \ EBR-Adresse (LBA) in die 2Variable
      2dup EBR 2! getLBAsect-hd#         \ EBR legen; LBA-Sektor nach sectbuf
      if
        cr ." HD-Fehler!" -1 exit        \ HD-Fehler? Dann Meldung und
      then                               \ Ausstieg mit fl=ffff. Sonst:
```

```
768         tmp1 @ drv# @ =                   \ Wenn tmp1 die eingegebene Nummer
769         if                                \ des Laufwerks drv# erreicht hat,
770           -1                              \ dann raus (-1 fuer until) mit fl=0.
771         else                              \ Sonst:
772           2 getLBAsect# or 0 =            \ Naechster EBR-Offset = 0 ? Ja, dann
773           if cr ." Letztes logisches Laufwerk = "
774           tmp1 @ . ." oder EBR-Fehler (?)"
775           -1 exit                         \ Fehlermeldung und raus mit fl=ffff.
776           then                            \ Nein, dann
777           tmp1 @ 1 + tmp1 ! 0             \ EBR-Zaehler um 1 weiterschalten und
778         then                              \ weiter in Schleife mit 0 fuer until.
779       until
780       0 ;
781
782   \ Das folgende Forth-Wort getbootext-hd# legt den Bootsektor des logischen
783   \ Laufwerks drv# der Festplatte hd# in den Sektorpuffer sectbuf und die
784   \ LBA-Nummer dieses Bootsektors in die 2Variable boot. Dazu wird erst ueber
785   \ getEBR-hd# der EBR geholt, dann aus dem EBR im Puffer sectbuf unter Offset
786   \ 1c6 der Abstand (in Sektoren) des gesuchten Bootsektors vom EBR und
787   \ schliesslich aus der 2Variablen EBR die LBA-Nummer des EBRs, die zum Abstand
788   \ addiert wird. Das Wort ist gegen vergessenes hd# oder/und drv# abgesichert.
789
790   : getbootext-hd# ( hd# drv# -- fl )     \ fl=0, wenn alles OK
791       2dup drv# ! hd# !                   \ drv# und hd# aufbewahren und
792       getEBR-hd#                          \ EBR von Laufwerk drv# holen
793       1 getLBAsect# EBR 2@ d+ boot 2!     \ LBA-Nummer des Bootsektors nach boot
794       if -1 exit then                     \ EBR-Fehler? Dann raus mit fl=ffff.
795       hd# @ boot 2@ getLBAsect-hd#        \ Sonst Bootsektor nach sectbuf
796       if cr ." HD-Fehler! " -1 exit then  \ HD-Fehler? Dann Meldung und Ausstieg
797       0 ;                                 \ mit fl=ffff; sonst: fl=0
798
```

Einnahmenüberschussrechnung in €

Forth Gesellschaft e.V.
Postfach 32 01 24
68273 Mannheim
01.01.2008 - 31.12.2008

Ideeller Bereich

Einnahmen aus ideellem Bereich

02110 Echte Mitgliedsbeiträge bis 256 Euro	2.296,00	
Beiträge		**2.296,00**
00870 Durchlaufende Posten Einnahmen	144,00	
Beiträge folgendes Jahr		**144,00**
02413 Überschuss Forth-Tagungen	361,64	
03220 Erhaltene Spenden / Zuwendungen	20,46	
Spenden		**382,10**
Summe Einnahmen aus ideellem Bereich		**2.822,10**

Kosten ideeller Bereich

02700 Kosten der Mitgliederverwaltung	-315,50	
02701 Büromaterial	-2,49	
02702 Porto, Telefon	-41,45	
02703 Einzugskosten	-8,50	
Kosten der Mitgliederpflege		**-367,94**
00216 Mikrocontrollerverleih	-491,95	
02704 Sonstige Kosten	-85,18	
02802 Geschenke, Jubiläen, Ehrungen	-54,00	
02811 Internetkosten	-532,13	
Sonstige Kosten ideeller Bereich		**-1.163,26**
Summe Kosten ideeller Bereich		**-1.531,20**
Summe Ideeller Bereich		**1.290,90**

Vermögensverwaltung

Einnahmen der Vermögensverwaltung

04150 Zinserträge 0 % USt	573,21	
Einnahmen der Vermögensverwaltung		**573,21**
03215 Sonstige Einnahmen	0,01	
Sonstige Einnahmen		**0,01**
Summe Einnahmen der Vermögensverwaltung		**573,22**

Kosten der Vermögensverwaltung

04712 Nebenkosten des Geldverkehrs	-163,45
Kosten Vermögensverwaltung	**-163,45**

Summe Kosten Vermögensverwaltung -163,45

Summe Vermögensverwaltung 409,77

Zweckbetrieb

Einnahmen aus Zweckbetrieben

06006 Einnahmen Zeitschrift VD 7%	1.828,58
Zeitschrift Vierte Dimension	**1.828,58**
01845 Umsatzsteuer 7 %	128,42
01910 Sammelkonto USt-Vorauszahlung/-Erstattung	176,75
Umsatzsteuer	**305,17**

Summe Einnahmen aus Zweckbetrieben 2.133,75

Kosten des Zweckbetriebes

06180 Aufwendungen für bezogene Leistungen	-1.735,94
06303 Transportkosten	-983,40
06304 Verpackungskosten Zeitschrift	-29,36
Zeitschrift Vierte Dimension	**-2.748,70**
06340 Verwaltungskosten	-646,00
Sonstige Kosten Zweckbetrieb	**-646,00**
00780 Abziehbare Vorsteuer 19 %	-401,93
Vorsteuerabzug	**-401,93**

Summe Kosten des Zweckbetriebes -3.796,63

Summe Zweckbetrieb -1.662,88

Überschuss 37,79

Vereinsvermögen Stand 31.12.2008

00940 Postbank	206,11
00950 Postbank Business Festgeld	18.108,55
09000 Saldovorträge Sachkonten	-18.276,87
Vermögenszuwachs	**37,79**

Protokoll der Mitgliederversammlung der Forth–Gesellschaft e.V. am 29. März 2009

Versammlungsleiter:	Martin Bitter
Schriftführer:	Anton Ertl
Beginn der Versammlung:	9h40
Feststellung der Beschlussfähigkeit:	20 Mitglieder anwesend

Bericht des Direktoriums

Ewald Rieger:

berichtet über das Wirtschaftsjahr 2008

Mitgliederentwicklung: mind. 122 Mitglieder; Altersstruktur

Gemeinnützigkeit:

Was ist Gemeinnützigkeit? Welche Teile sind steuerpflichtig?

Kassenbericht

Erich Wälde fragt, wie das Guthaben zustande kam. Antwort: Mitgliedsbeitrag war auf höhere VD-Kosten ausgelegt, und die Kosten gingen durch ehrenamtliche Herausgeberschaft zurück, die Mitgliedsbeiträge erst später.

Ulrich Hoffmann:

berichtet über die VD. Michael Kalus fragt nach der Produktion, und sie wird erklärt.

berichtet über den Server der Forth-Gesellschaft: 284 Tage uptime. Keine Einbrüche.

Bernd Paysan:

Projekte: Mac Mini wurde angeschafft, um Gforth und BigForth auf MacOS X zu portieren bzw. die Portierung zu verbessern.

FPGA-Board: Erweiterung Richtung Ethernet.

Lego-NXT: läuft noch nicht so recht.

Außendarstellung:

Anmeldung für Linuxtag erfolgt. Diskussion zu Hinguckern.

Bericht der Kassenprüfer (Michael Kalus, Carsten Strotmann)

Tippfehler gefunden, sonst kein Fehler.

Entlastung des Kassenführers:

19J, 0N, 1E.

Entlastung des Direktoriums:

einstimmig entlastet.

Wahl des neuen Direktoriums:

Das alte Direktorium ist bereit, die Geschäfte weiterzuführen.

Es gab keine Vorschläge für andere Direktoren.

Das Direktorium wurde einstimmig wiedergewählt.

Projekte:

Bernd Paysan schlägt Gforth EC für AVR vor; 4 AVR–Boards werden benötigt.

Diskussion zu Schulungen auf Vorschlag von Erich Wälde. Tutorenschulungen werden von Michael Kalus und Carsten Strotmann koordiniert.

Klaus Schleisiek fragt nach Texten zum Forth-Lernen: Antworten: Starting Forth, Programming Forth (Stephen Pelc), Gforth-Tutorial, Zech-Buch für System-Bauen.

Gforth EC Dokumentation: Gerd Franzkowiak meldet sich.

Kochbuch: Ansprechpartner Erich Wälde

Gerd Franzkowiak fragt wegen Gforth für Buildroot etc. an (cross-compilierbares Gforth)

Forth für PIC18Fxxxx (Carsten Roederer) gibt's laut Adolf Krüger.

Verschiedenes:

Nächste Forth-Tagung

Niederländer wollen leider nicht.

Anflanschen an eine Tagung? Embedded Systems?

Ein halber Freiwilliger für Rostock.

Ankündigen der Forth-Tagung in der Presse.

Ende der Sitzung:

12h07

Forth–Gruppen regional

Mannheim
Thomas Prinz
Tel.: (0 62 71) – 28 30 (p)
Ewald Rieger
Tel.: (0 62 39) – 92 01 85 (p)
Treffen: jeden 1. Dienstag im Monat
Vereinslokal Segelverein Mannheim e.V. Flugplatz Mannheim–Neuostheim

München
Bernd Paysan
Tel.: (0 89) – 79 85 57
bernd.paysan@gmx.de
Treffen: Jeden 4. Mittwoch im Monat um 19:00, im Chilli Asia Dachauer Str. 151, 80335 München.

Hamburg
Küstenforth
Klaus Schleisiek
Tel.: (0 40) – 37 50 08 03 (g)
kschleisiek@send.de
Treffen 1 Mal im Quartal
Ort und Zeit nach Vereinbarung (bitte erfragen)

Mainz
Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rowila@t-online.de

Gruppengründungen, Kontakte

Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

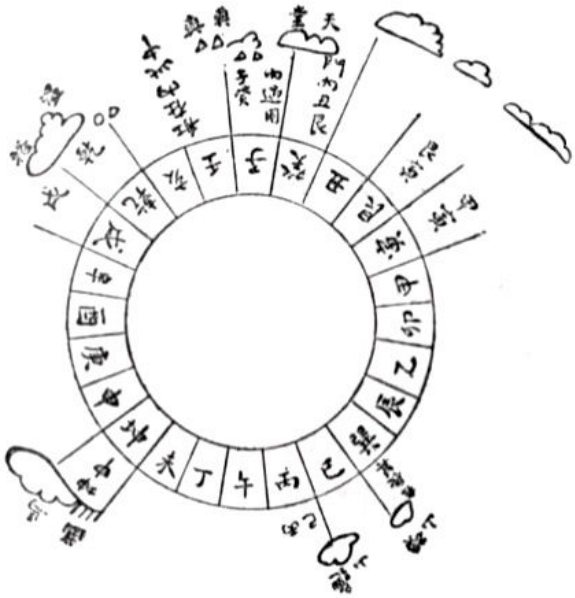
Carsten Strotmann
microcontrollerverleih@forth-ev.de
mcv@forth-ev.de

Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)
Klaus Schleisiek–Kern
Tel.: **(0 40) – 37 50 08 03** (g)

KI, Object Oriented Forth, Sicherheitskritische Systeme
Ulrich Hoffmann
Tel.: **(0 43 51) – 71 22 17** (p)
Fax: **– 71 22 16**

Forth–Vertrieb
volksFORTH
ultraFORTH
RTX / FG / Super8
KK–FORTH
Ingenieurbüro
Klaus Kohl–Schöpe
Tel.: **(0 70 44)–90 87 89** (p)



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Einladung zur

Forth–Tagung 2010

vom 26. bis 28. März 2010

im Strand–Hotel Hübner

Seestraße 12, 18119 Rostock-Warnemünde

http://www.hotel-huebner.de

geplantes Programm

Donnerstag, 25.03.2010
nachmittags Treffen der Frühankommer
Forth–200x–Standard–Treffen

Freitag, 26.03.2010
nachmittags Beginn der Forth–Tagung
Vorträge und Workshops

Samstag, 27.03.2010
vormittags Vorträge und Workshops
nachmittags Exkursion

Sonntag, 28.03.2010
09:00 Uhr Mitgliederversammlung
nachmittags Ende der Tagung

Anreise siehe: http://www.hotel-huebner.de/kontakt/anfahrt.html

Quellen: www.hotel-huebner.de