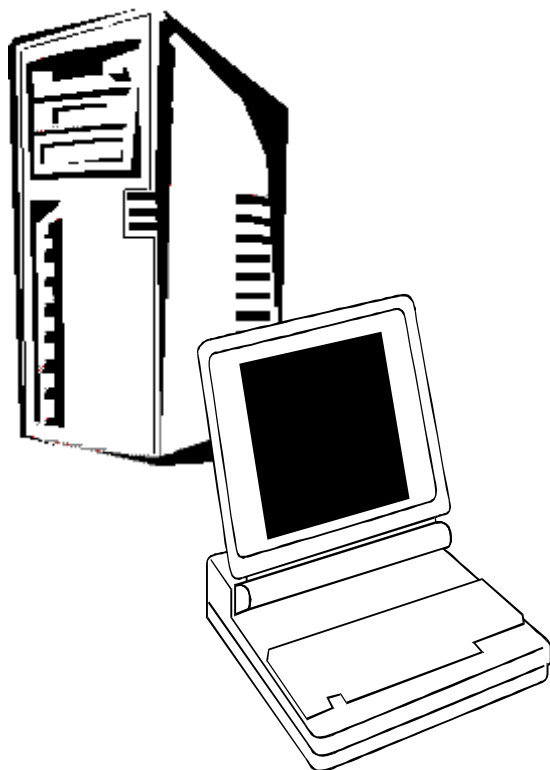
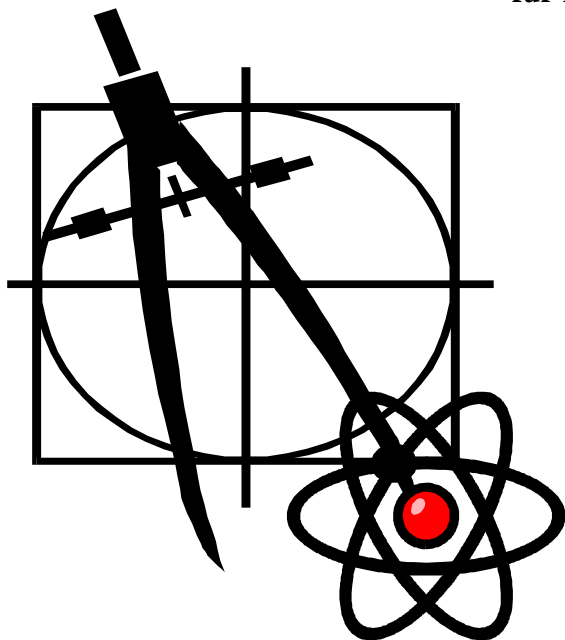


VIERTE DIMENSION

Das FORTH-Magazin

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



In dieser Ausgabe:

Leserbriefe und Rezensionen

Leser schreiben, was sie interessiert

Hashing, Teil 2

Fortsetzung und Schluß des Aufsatzes über das Hashing im ZF

Codeknacker

Einen Aufsatz über polyalphabetische Verschiebungen

CFA2NAME

Modifikationen am F-PC

Körperschallanalyse

Vortrag von J. Reilhofer in Oberammergau

Reed-Solomon, Teil 2

Fortsetzung und Schluß des Aufsatzes über Fehlerkorrekturen

Spaß mit Forth

6 Autoren aus 4 Ländern beschreiben, warum sie als Hobbyisten mit Forth „arbeiten“

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

FORTH - Shirt



Räumungsverkauf
T - Shirt: hellgrau / grün
in Größe M-L-XL **15 DM**
Sweat-Shirt: grau / grün
in Größe M-L-XL **25 DM**
(+ Porto)

ForthWORKS

Ulrike Schnitter
Nelkenstr. 52
85716 Unterschleißheim
fon/fax 089-310 33 85

Hier könnte IHRE Anzeige stehen

Setzen Sie sich doch einfach einmal mit dem
Büro der Forthgesellschaft e.V. in Verbindung.

Dipl.-Ing. Arndt Klingenberg

Tel.: ++32 +87 -63 09 89 (Fax: -63 09 88)
Waldring 23, B-4730 Hauset, Belgien
akg@.forth-ev.de

Computergestützte Meßtechnik und Qualitätskontrolle, Fuzzy, Datalogger, Elektroakustik (HiFi), MusiCassette HighSpeedDuplicating, Tonband, (engl.) Dokumentationen und Bedienungsanleitungen

Forth Engineering Dr. Wolf Wejgaard

Tel.: +41 41 377 3774 - Fax: +41 41 377 4774
Neuhöflirain 10
CH-6045 Meggen <http://holonforth.com>

Wir konzentrieren uns auf Forschung und Weiterentwicklung des Forth-Prinzips und offerieren HolonForth, ein interaktives Forth Cross-Entwicklungssystem mit ungewöhnlichen Eigenschaften. HolonForth ist erhältlich für 80x86, 68HC11 und 68300 Zielprozessoren.

KIMA Echtzeitsysteme GmbH

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

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

Ingenieurbüro Dipl.-Ing. Wolfgang Allinger

Tel.: (+Fax) 0+212-66811
Brander Weg 6
D-42699 Solingen

Entwicklung von µC, HW+SW, Embedded Controller, Echtzeitsysteme 1-60 Computer, Forth+Assembler PC / 8031 / 80C166 / RTX 2000 / Z80 ... für extreme Einsatzbedingungen in Walzwerken, KKW, Medizin, Verkehr / >20 Jahre Erfahrung.

FORTECH Software Entwicklungsbüro Dr.-Ing. Egmont Woitzel

Joachim-Jungius-Straße 9 D-18059 Rostock
Tel.: (0381) 405 94 72 Fax: (0381) 405 94 71

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.

Ingenieurbüro Klaus Kohl

Tel.: 08233-30 524 Fax: —9971
Postfach 1173
D-86404 Mering

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

Impressum	4
Editorial	4
Leserbriefe	5
Was Sie uns und den Lesern der Vierten Dimension mitteilen wollten	
Briefe aus der FIG Silicon Valley	6
<i>Henry Vinerts</i>	
„Körperschallanalyse“ - Möglichkeiten der Schadensfrüherkennung unter dynamischen Bedingungen, J. Reilhofer	9
CFA2NAME	12
Modifikationen an F-PC, <i>Wolfgang Allinger</i>	
HASHING, Teil 2	15
Fortsetzung des Aufsatzes aus der VD 04/99, <i>Friederich Prinz</i>	
Codeknacker, Teil 1	22
Ein Programm zum Knacken von polyalphabetischen Codes, <i>Hugh Aguilar</i>	
Spaß mit Forth	30
<i>Aguilar, Jakeman, Ouwerkerk, Prinz, Bitter, Behringer</i>	
Reed-Solomon-Fehlerkorrektur, Teil 2	34
<i>Glenn Dixon</i>	
Einladung und Tagesordnung zur Jahresversammlung der Forthgesellschaft e.V.	38

In der nächsten Ausgabe finden Sie voraussichtlich:

- JMPs > 32 k unter DOS, von Fred Behringer
- ausstehende Besprechungen unserer ‚Schwesterjournale‘
- Codeknacker, Teil 2, von Hugh Aguilar
- Farbig Drucken mit WIN32FOR, von Martin Bitter

IMPRESSUM

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth-Gesellschaft e.V.

Postfach 16 12 04

D-18025 Rostock

Tel.: 0381-400 78 28

E-Mail:

SECRETARY@FORTH-EV.DE

DIREKTORIUM@FORTH-EV.DE

Bankverbindung: Postbank Hamburg

BLZ 200 100 20

Kto 563 211 208

Redaktion & Layout

Friederich Prinz

Hombergerstraße 335

47443 Moers

Tel./Fax.: 02841-58 3 98

E-Mail:

VD@FORTH-EV.DE

FRIEDERICH.PRINZ@T-ONLINE.DE

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluß 1999

März, Juni, September, Dezember

jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

DM 10,- zzgl. Porto u. Verp.

Manuskripte und Rechte

Berücksichtigt werden alle eingesandten Manuskripte. Leserbriefe können ohne Rücksprache gekürzt wiedergegeben werden. Für die mit dem Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in diesem Magazin veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, Nachdruck sowie Speicherung auf beliebigen Medien ist auszugswise nur mit genauer Quellenangabe erlaubt. Die eingereichten Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen - soweit nichts anderes vermerkt ist - in die Public Domain über. Für Fehler im Text, in Schaltbildern, Aufbauskizzen u.ä., die zum Nichtfunktionieren oder eventuellem Schadhafwerden von Bauelementen oder Geräten führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes. Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.



Liebe Leser,

der Wechsel in das Jahr 2000 ist vollzogen, die Welt ist nicht untergegangen – auch nicht die sogenannte zivilisierte Welt. Es sind keine Kraftwerke explodiert und keine Flugzeuge abgestürzt und auf meinem Girokonto sind weder unerwartete Millionen aufgetaucht, noch ein nicht nachvollziehbares Sal-

do. Leichtsinzig wie wir älteren Leute nun mal sind, haben meine Frau und ich genau darauf vertraut – und nicht einen Groschen in die unzähligen „Y2K Präventionen“ investiert. Das dabei gesparte Geld habe ich persönlich lieber in einen Vorrat Zigarren investiert, natürlich nicht über das gewöhnliche Feiertagsmaß hinaus, weil die Dinger nicht mehr richtig schmecken, wenn sie zu lange aus der Klimakammer heraus sind. Ich habe es also vorgezogen, mir selbst „einen blauen Dunst“ vorzumachen.

Bei dem Thema „blauer Dunst“ frage ich mich unwillkürlich, ob es nicht vielleicht besser gewesen wäre, wenn die „zivilisierte“ Welt wenigstens ein bißchen gebebt hätte. Vielleicht hätte dann jemand gemerkt, daß der Wechsel in das nächste Jahrtausend erst noch bevorsteht. Fehlende Fertigkeiten in der Mathematik werden in unserer Gesellschaft nur in Ausnahmefällen als Manko angesehen. Daß es bei den allermeisten unserer Zeitgenossen offenbar nicht einmal zum korrekten Zählen reicht, muß schon entsetzen. Es erscheint mir notwendiger denn je, daß sich die Menschen zukünftig weniger mit „vernetztem Denken“ und Arbeitsteilung im Sinne der Delegation von eigenen Verantwortungen befassen, als vielmehr wieder auf „Wissenskerne“ besinnen und konzentrieren. Im übertragenen Sinne würde ich formulieren wollen, daß wir allesamt einige Schritte von den „eierlegenden Wollmilchsaucopilern“ zu überschaubaren und handhabbaren (forthigen) Systemen zurück machen müssen.

Ich persönlich mag mir weder das Zählen abnehmen lassen, noch die Verantwortung für meine Zählweise delegieren. Ich empfinde es als wenig befriedigend, bei einer fehlerhaften, von mir erstellten Routine auf die Hersteller von Betriebssystem und Compiler zu zeigen und darauf zu verweisen, daß ich das Zählen 'Denen' überlassen habe. Ebenso unbefriedigend erscheint es mir, den Jahrtausendwechsel zweimal feiern zu müssen, weil sich der Groß- und Einzelhandel bedauerlicher Weise verzählt haben. Ich bin aber ganz sicher, daß dem Handel sein „Irrtum“ rechtzeitig vor dem nächsten Jahreswechsel bewußt wird. Und ich warte mit großem Interesse darauf, wem zuerst einfällt, daß der Gregorianische Kalender in bezug auf Christi Geburtsdatum durchaus einige Jahre Spielraum läßt. Ein Schelm wer Böses dabei denkt...

Natürlich betrifft die Forther weder der „blaue Dunst“, noch trifft sie der Vorwurf, nicht einmal zählen zu können. Forther können quasi „von Natur aus“ zählen, und – was mir viel wichtiger erscheint – man darf auf die Forther zählen. Den Aufruf, dabei zu helfen, die VD webtauglich zu machen, haben gleich mehr als eine handvoll Forther ernst genommen und ihre tatkräftige Hilfe angeboten. Ihnen allen soll an dieser Stelle im Namen der Forthgesellschaft und der VD herzlich gedankt sein !

Die Ausgaben 01/98 bis 04/99 können Sie bereits von der Homepage der Forthgesellschaft aus erreichen. Die kommenden Ausgaben werden zukünftig mit ca. 4 Wochen Verzögerung ebenfalls dort eingestellt werden. Besuchen Sie doch gelegentlich **WWW.FORTH-EV.DE** und schreiben Sie uns, wie Ihnen die VD im WEB gefällt.

Friederich Prinz



Quelltext Service

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können diese Texte auch direkt bei uns anfordern und sich zum Beispiel per E-Mail schicken lassen. Schreiben Sie dazu einfach eine E-Mail an die Redaktionsadresse.

fep



Lieber Friederich Prinz

Zu VD 4-99

Hashing

Ich erinnere mich dunkel, daß mit Hashing einmal etwas war, vor vielleicht 12 Jahren, weiß aber nicht, war es in der VD oder im Arbeitskreis Forth, den ich einige Jahre an der VHS Darmstadt moderierte. Mit der Ahnung einer Erinnerung fand ich auf Anhieb:

Niklaus Wirth: Algorithmen und Datenstrukturen, Teubner 1983, ISBN 3-514-02250-8, S. 296 - 306. (Programmbeispiele in Pascal)

Zum Logo, S. 32

Nr. 2, 3 und 4 werden nur von Kennern verstanden = ungünstig; Nr. 1 versteht jeder, was wichtiger ist als originell zu sein. Mir ist nichts besonders Originelles eingefallen, aber ich meine, es müßte noch etwas Forth-typisches dabei sein, so in der Richtung

Forth-Gesellschaft e.V.

: „FORTH kann fast alles“

Das von M. Pilot erwünschte Finanzspiel wäre interessant. Man könnte es ja gleich Finanzpilot nennen. Im Arbeitskreis FORTH war kurzzeitig ein Wirtschafts-Professor, der nach einer passenden Programmiersprache suchte, mit der man Operationen mit größeren Determinanten möglichst einfach durchführen kann. Er kam aber bald wieder von Forth ab. Ich wollte mal nachfragen, wie das heute gemacht wird, weil ich gerade seine Telefon-Nummer fand; aber er ist vor 5 Jahren ausgezogen.

Schon vor Jahren schwebte mir eine sciencefictionale Wirtschaftsglosse vor, nur habe ich nicht genügend Fachkenntnis, um sie auszuführen. Also nur ein Stenogramm:

Aktien!

„Legen Sie Ihr Geld gewinnbringend in Aktien an! Wenn sie es geschickt genug anlegen, haben sie bald mehr, als wenn sie arbeiten!“ Immer mehr Menschen folgen diesem Motto – es gibt immer weniger Arbeitslose.

Die beliebteste Automarke kann nicht liefern, die Aktien fallen. Die Zulieferer haben nicht mehr genügend gute Arbeitskräfte, das Material kommt nicht termingemäß, die letzte Serie war Ausschuß. Die Lawine fängt an zu rutschen – Währungsverfall. Viele werden arm.

Aber einige Schlaue haben es geahnt, und rechtzeitig für ihre Aktien Reales gekauft oder an einem lebenswichtigen Betrieb die Aktienmehrheit erworben, um dann neu zu starten.

Andreas Soeder

Hallo Friederich!

Ich habe gerade Deinen Artikel in der VD 4/99 zum Thema Hashing überflogen. Im wesentlichen beschreibst Du da Hash-Funktionen, die ich gern als die „normalen“ oder „Wald-und-Wiesen“-Hash-Funktionen nenne. Dies ist keinesfalls abwer-

tend gemeint (es ist insbesondere kein von Fachleuten anerkannter Begriff :), denn diese Klasse der Hash-Funktionen versteht stillschweigend und zuverlässig seit Anbeginn des Computer-Zeitalters ihren Dienst.

Leider hast Du den Text von Ray Duncan nur erwähnt (ich würde mich freuen, wenn ich eine Kopie davon irgendwo bekommen könnte), denn er deutet eine ganz andere Klasse von Funktionen an, die wirklich spannend ist, die Klasse der **perfekten Hash-Funktionen** (und diesmal ist es ein Fachbegriff). Also:

$n = h(x)$ mit Kollisionen: „normale“ Hash-Funktion

$n1 = h(x1), n2 = h(x2) \quad n1 > n2, \text{ wenn } x1 > x2:$
„geordnete“ Hash-F.

$n1 = h(x1), n2 = h(x2) \quad x1 \neq x2 \implies n1 \neq n2:$
„perfekte“ Hash-F.

(Anzahl der n's größer als die der x's)

$n1 = h(x1), n2 = h(x2) \quad x1 \neq x2 \implies n1 \neq n2,$
Anzahl der n's =

Anzahl der x's:

„minimal perfekte“ Hash-F.

Legende: n ist der „Hash-Wert“ von x, $n = h(x)$

Den Nobelpreis für Mathematik gibt es, wenn Du ein garantiert erfolgreiches Verfahren findest, das geordnete, minimal-perfekte Hash-Funktionen liefert. Reich wirst Du, wenn Du dieses Teil an Oracle & Co. verkaufen kannst... :)

Für "perfekte" Hash-Funktionen, die manchmal auch minimal sind, kannst Du Dir mal das GNU-Tool "gperf" ansehen. (Da ist auch eine Literaturliste zum Thema drangeheftet...) Ich kenne einen attraktiven Algorithmus, der bis zu ca. 2 Millionen Schlüssel perfekte Hash-Funktionen liefern kann, nur sind die Tabellen solcher Verfahren meist statisch und für Forth-Dictionaries daher nicht geeignet. Du kannst obige Zusammenstellung aber noch ergänzen, wenn Du die spärlich dokumentierte Klasse der dynamisch-perfekten Funktionen aufnimmst. Dann sind wir aber so tief in der Zahlentheorie eingetaucht, daß mir speiübel wird...

Also, wenn Du mir sagen könntest, wo ich Ray Duncans Text herbekommen kann (oder wenn Du ihn mir irgendwie schicken kannst), dann wäre ich glücklich wie ein Stinktier im Wäschekorb! Echt!

Gruß,

Olaf Stoyke

Leider kann ich bezüglich des Textes von Ray Duncan nicht helfen. Der Aufsatz über das Hashing lag, wie bereits in der VD erwähnt, schon etwas länger hier herum. Die Originaldokumente habe ich nicht mehr. Ich hoffe, daß der Johannes Teich weiterhelfen kann und zeitlich „tiefer“ archiviert als ich es tue. Dafür würde mich die Literaturliste interessieren, die von Dir erwähnt wird. Vielleicht ist Jemand so freundlich und schickt mir die zu ?

fep



Leserbriefe

Lieber Fritz, lieber Martin,

Die VD ist eingetroffen. Ausgezeichnet! Da steckt ja eine ungeheure Arbeit drin. Bei den vielen redaktionellen Kommentaren, die das Ganze zusammenschweißen und der Zeitschrift ein einheitliches Gesicht geben. Da arbeitet doch sicher die gesamte Moerser Gruppe mit?

Oder „reibst“ Du Dich allein „auf“ ? Es wird zukünftigen Direktorien schwerfallen, Gelder für die VD-Redaktion zu bewilligen. Sie werden in aller Ewigkeit auf Dich und Deine Mitarbeiter verweisen und nach ebensolchen „kostenneutralen“ Enthusiasten Ausschau halten. - Ich lese die VD nicht, wie Joerg Pohl, von hinten nach vorn. Aber beinahe. Erst hinten, dann ganz vorn, die Leserschriften und Kurzmeldungen - und dann die Hauptartikel aus der Heftmitte. Ganz richtig durchlesen tue ich die Artikel aber meist immer erst dann, wenn ich etwas Bestimmtes suche. Und das dann unabhängig von Raum und Zeit. Momentan waren es die Jahrgänge 1990/91. Interessant, welche Autoren damals eine „fleißige Ader“ hatten. - Die Reaktionen auf das Rätsel in der diesmaligen Ausgabe waren nicht schlecht. Sollte man weiterführen! Henry Vinerts Formulierung war, glaube ich, die beste.

Fred Behringer

Anmerkung: Die meiste Arbeit muß ich schon selbst tun. Das ist sozusagen zwangsläufig so. Wir sehen keinen Bedarf an ewig langen „Redaktionsbesprechungen“ über Inhalte und Themen. Die Autoren und die Leser geben uns vor, was in der jeweils nächsten Ausgabe stehen soll. Das „Montieren“ der Texte läßt sich am einfachsten dadurch erledigen, daß man sich an den PC setzt und konzentriert arbeitet – allein. Bei vielen anderen Dingen entlasten mich die Kameraden hier in Moers aber nach Kräften. So hat z.B. Martin Bitter dankenswerter Weise die Besprechungen der 'Embedded' übernommen. Wer den Martin kennt, weiß, daß er diese Arbeit sehr ernst nimmt. Wenn es um das Aufarbeiten von Bildern und Graphiken geht, dann hilft meist Michael Major kräftig aus, mit Rat und Tat. Und das Korrekturlesen nimmt mir meine Frau ab. Gerade diese Arbeit fällt mir selbst eher schwer.

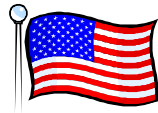
Zusammenfassend: Nein, ich muß mich nicht „aufreiben“. Ich darf viele interessante Beiträge als Erster lesen.

fep

M. Pilot hat einen interessanten Appell an die Programmierer gerichtet, sich mit Wirtschaftsspielen zu beschäftigen. Entgegen dem, was mancher denken könnte, hat das, was unser Freund Pilot meint, nichts mit Spielen im eigentlichen Sinn zu tun, höchstens im Ausnahmefall. Es handelt sich um Entscheidungen, die gegeneinander (Konkurrenz) oder (teilweise) auch miteinander (Kooperationsvertrag) gefällt werden und möglichst optimal ausfallen sollen. M. Pilots Bemerkung über die Programmierer, diese „fixen Kerlchen“ (à la „Ich kenne da einen Rechtsanwalt, der wird das überprüfen“ aus „A Streetcar Named Desire“), ist sicher nicht ernst zu nehmen. Ganz bestimmt hat er eigene Ideen und ist zu Zusammenarbeit bereit. Ich meinerseits habe mit Fritz gespro-

chen. „Forth und die Spieltheorie“ in den Grundzügen darzustellen, würde mich reizen. Nicht für sofort. Aber für unmittelbar danach.

Fred Behringer



Lieber Friederich,

auf dem Heimweg am Nachmittag nach dem SVFIG-Treffen letzte Woche dachte ich, daß es zu schade ist, daß ich das Treffen im nächsten Monat verpassen muß (Ich werde an einem Federballturnier in Florida teilnehmen) und nicht in der Lage sein werde, Dir einen Bericht darüber zu geben. Doch dann erkannte ich plötzlich, daß ich Dir wahrscheinlich einen halben Bericht jeden Monat liefern kann, ohne überhaupt selbst zu den Treffen zu gehen. Falls ich schreiben würde, daß „Zu Beginn des Treffen um 10 Uhr um die 10 bis 15 Leute da waren, sich die Anzahl auf ungefähr 20 im Laufe des Tages erhöhte, mit einigem Wechsel, and Dr. Ting die ersten zwei Stunden über seine gegenwärtigen Projekte sprach“, dann würde das wahrscheinlich so ziemlich das ganze Jahr gut sein.

Und so war es. Ich denke nicht, daß wir ohne Dr. Ting ganztägige Treffen (10 bis 16 Uhr) durchführen würden. Er genießt es sichtlich, seine Leidenschaft für Forth mit uns zu teilen. Dieses mal hörten wir etwas über sein Projekt, seinen eigenen Computer mit dem Xilinx Board zu „bauen“, unter Nutzung von eForth, welches mit der Hilfe von FPC compiliert wurde. Danach kamen einige Bemerkungen über den Drachen der über Taiwan hereinbrach (es ging um das kürzliche Erdbeben) und Preissteigerungen bei DRAM's und anderer Computer-Hardware auslöste. Darauf folgte ein Weiteres seiner Projekte: einen Zeitablauf des 20. Jahrhunderts als Poster herzustellen, der die wichtigsten Ereignisse in den Computerwissenschaften, in Geschichte, Politik, allgemeine Wissenschaften und, natürlich, Forth enthält. Wie ich schon früher sagte, ich glaube nicht, daß dieser Mann jemals schläft.

John Hall brachte ein paar Apple iBook Computer mit, um sie uns am Nachmittag zu zeigen. Einer war weiß-blau, der andere weiß-orange, und sie konnten miteinander per Funk kommunizieren. Sie sahen wie tragbare Toilettensitze für Kinder mit Henkel aus, kosten aber etwas mehr - 1499 \$ (das ist deutlich unter 1 1/2 Tausend). Wenn jemand in der Lage ist beim Systemstart in Open Firmware zu gelangen, dann erhält man durch Eingabe von „kudos“ die Namen der Entwickler, von denen John Hall einer ist und Ron Hochsprung (der auch zum Treffen kam) ein anderer.

Soviel zum SVFIG-Treffen. Ich möchte noch erwähnen, daß ich am folgenden Dienstag einen halben Tag frei nahm und zur Embedded-Systems-Konferenz in San José ging. Sie scheint mehr Platz in jedem Jahr einzunehmen, es waren über 300 Aussteller dieses Jahr da. Zeitweise scheint es, daß mehr Teilnehmer von den verschiedenen Firmen als Besucher anwesend sind. Es gab keine Präsentation von Forth in diesem Jahr, wenn man nicht Patriot's ShBoom zählt. WINCE war überall, Java auch, begleitet von soviel Debug-Lösungen wie Kopfschmerzpillen in der Apotheke. Ich glaube, daß ich kurz Klaus Flesch in einem roten T-Shirt sah, aber er war zu



schnell für mich, um ihn zu erreichen.

Der Höhepunkt der Show, für mich und eine Reihe anderer Forther, die gekommen waren, um ihn zu hören, war Cliff Stoll. Er hielt eine sehr lebendige und unterhaltsame Rede und entließ die Zuhörer mit der Herausforderung seine Skepsis über Computer, insbesondere in der Ausbildung von Kindern, zu widerlegen. Cliff, der einen Doktorgrad in Astronomie besitzt, ist für seinen Erfolg bei Entlarfung eines Hackers aus Übersee bekannt, der vor 10 Jahren versucht hatte, Geheimnisse aus den Lawrence Berkeley Laboratorien zu stehlen. Ich weiß, daß dieser Brief Eure Zeitschrift nicht mehr rechtzeitig erreicht, aber ich möchte Dir (und Du kannst es an die anderen weiterleiten) ein SEHR FROHES OKTOBERFEST wünschen !

Henry

Hallo, Friederich,

da ich das Oktober-Treffen der SVFIG verpaßte scheint es eine lange Zeit zu sein, seit ich Dir das letzte Mal schrieb; jedoch kommt diese Notiz eher als üblich, da das November-Treffen am zweiten Samstag des Monats stattfand -- die FORML-Konferenz und Thanksgiving hatten an den folgenden beiden Wochenenden Vorrang.

Wieder zeigten sich, wie üblich, um die 20 Leute in mittlerem und höherem Alter im Verlauf des Tages bei unserem Treffen. Wir vermissten möglicherweise nur ein paar der älteren, die wohl zur „Hackers“-Konferenz abgewandert waren, die am gleichen Wochenende stattfand.

Sehr zu meiner persönlichen Freude haben sich John Rible's Vorlesungen über CPU-Design zu grundlegenden und verständlichen Inhalten verschoben. Tatsächlich gingen wir dieses Mal zum Stand der Technik vor 35 Jahren zurück. In den frühen Sechzigern hatten einige Wissenschaftler in den Bell Laboratorien ein Modell eines sehr einfachen Computers als Lernhilfe entwickelt. Dieses, CARDIAC genante, Modell bestand aus einem Umschlag und einigen Verschiebestreifen aus Pappe. Es wurde als Bausatz (wie bei den Papierpuppen) geliefert, zusammen mit einem kleinen Handbuch, welches die Arbeit eines „modernen“ Computers erklärte, der Elektrizität, Vakuumröhren, Lochkarten und Ferritkern-Speicher nutzte. Die National Science Foundation verteilte dieses Kit an leistungsfähige Studenten in den nationalen Hochschulen. John Rible und Skip Carter waren unter diesen Studenten.

Jetzt haben John und Skip die Copyright Erlaubnis erhalten, neue Modelle des CARDIAC als Lernhilfe herzustellen und zu modifizieren. John ist auf dem besten Wege mit seinem CARDIAC 2000. Der ursprüngliche CARDIAC hatte nur 10 Opcodes und 100 Speicherzellen, von denen jede drei dezimale Stellen beinhaltete. Etwa 17 Eingabekarten bildeten den Bootstrap-Lader, der die Maschine startete und ein Additionsprogramm laden konnte. John hat die Opcodes überarbeitet und plant das Handbüchlein zu überarbeiten (insbesondere es zu modernisieren). Der nächste Schritt könnte darin bestehen, aus CARDIAC einen Stackcomputer zu machen und es als Lernwerkzeug für Forth zu nutzen. Aber dann wäre, als zusätzlicher Lernschritt, die Einführung hexadezimaler Zahlen notwendig.

Was mich am meisten beeindruckt, hat ist das BEFEHLS-HANDBUCH. Seit Jahren klage ich, daß ich noch kein Forth-System gefunden habe, mit dem ich lernen kann, da keines ein gutes Manual hat. Ebenso gab es nie gute Forth-Bücher mit einem sofort nutzbaren, einfachen System auf einer Diskette. Vielleicht ist es nie zu spät anzufangen, sogar wenn wir zurück zu Pappcomputern gehen müssen.

An diesem Punkt, dieses Bemühen betreffend, möchte ich an einige Gedanken erinnern, die Rafael Deliano in früheren Ausgaben der VD präsentierte:

1) auf Seite 23, VD 4/96 führt er nur Brodie und Zech als Bücher auf, die in Deutschland Eindruck machten. Ich habe beide Bücher von Brodie, aber habe niemals die von Zech gesehen. Wurden letztere ins Englische übersetzt? Kann jemand eine kurze Zusammenfassung darüber geben, was Zech's Bücher so gut macht. Wir müssen lernen, wie man gute Lehr- und Lernwerkzeuge herstellt.

2) auf Seite 26, VD 1/96 zitiert Rafael Jef Raskin's Bemerkungen über von Computerfirmen hergestellte Toaster, die Bedienanleitungen benötigen, die man mit dem Handwagen herumfahren muß. Wir wollen keine solchen Manuals haben, aber ich denke daß, dieser Artikel ein guter Kandidat für einen deutschen Nachdruck in der VD wäre, falls einer Eurer guten Übersetzer an etwas anderem als meinen SVFIG-Berichten arbeiten möchte.

Oh, ja! Dr. Ting war auf dem Treffen, wie üblich, und philosophierte über verschiedene Formen von Intelligenz: menschlicher, maschineller und künstlicher. Er sprach über Formgesetze, Turing-Automaten, NAND-Gatter und andere Konzepte, die in den fortwährenden Kurs gehören, den er CPU Design 101 nennt. (Es scheint, daß an amerikanischen Universitäten die meisten einführenden Kurse des Hauptstudiums, also die des dritten Studienjahres, mit 101 bezeichnet werden, wie z.B. Elementarmathematik für fortgeschrittene Studenten 101.) Darüber hinaus hörten wir mehr darüber wie eForth (Version 2.xx) sich zu einem kompletten, sich selbst genügenden Entwicklungssystem entwickelt, obwohl einige seiner Eigenschaften sich zu figForth zurückbewegen.

Es war einige Zeit für zukünftige FORML-Teilnehmer reserviert worden, um ihre Vorträge durchzusehen, aber keiner nutzte diese Gelegenheit. Es gab mehr von der immer populären Freizeit, die wir im geselligen Beisammensein und mit Gesprächen verbrachten, Gespräche über alle Themen, die spontan in einer solchen Gruppe wie der SVFIG aufkommen. Euer freundlicher Berichtersteller hatte sein Exemplar der VD 4/99 erhalten und konnte einige Leute daran interessieren über Georg's Glühlampen-Rätsel nachzudenken.

Einige Freiwillige kamen mit ihren eigenen Rätseln, was mich auf den Gedanken brachte, daß es keine schlechte Idee wäre, eine regelmäßige Rätselcke in jeder Ausgabe der VD zu haben.

Eine Sache noch heute: Ich fühlte mich ziemlich „gekitzelt“, daß ich eine Reihe hilfreicher Tips für unser Übersetzungsrätsel erhalten hatte, speziell dieses „Voll das Ver..schungsprogramm, ey!“ (Wessen Bein versucht Ihr auszureißen, hej?). Ich muß nach Hamminkeln und München schreiben, um mich dafür separat zu bedanken. In der Zwischen-



Leserbriefe

zeit, viel Glück an meinen freundlichen Übersetzer in Mittweida! (tb: "Thanks a lot, Henry! I enjoy it very much!")

Henry

Anmerkung: Wer faßt sich ein Herz, und dem Henry zusammen, was an Zech's Büchern so wertvoll ist ?

red

Die regelmäßige Rätsecke fänden wir – die Redaktion – auch nicht schlecht. Dazu müssen solche Rätsel aber auch gestellt werden. Henry, vielleicht schickst Du uns Rätsel Deiner Kameraden aus den USA herüber ?

red

Doch nicht so **einfach** oder **Gewohnheitsfrage** ?

Versucht man, einem X-länder zu erklären, warum die Sprache Y besser ist, so wird er, soweit er höflich und gebildet ist und neben X zumindest auch Y versteht, beipflichten: Y ist besser. Sprechen aber wird er weiterhin X. Das hat er schon als Kind gekonnt.

beh

Erste Reaktionen auf die VD im Netz !

Betreff: Re: Vierte Dimension im Netz

Egmont Woitzel@fortech.de said concerning „Vierte Dimension im Netz“ :

- > Nachdem wir Verschiedenes ausprobiert
- > haben, sind jetzt die ersten Ausgaben
- > der Vierten Dimension (1..3/98) unter
- > www.forth-ev.de/vd.htm zu finden. Die
- > übrigen Ausgaben folgen so bald als
- > möglich.

Das bemerkte ich, als ich plötzlich (nach so ca. 18 Jahren) von einer früheren Freundin aus München angerufen wurde. Sie hatte meine (sonst weitgehend unbekannte Tel. Nr.) von Forth-ev.de (Das kostete 60 min und demnach 10 Euro fürs Teflon). Das sind die netten Suchmaschinen im INTER. Wenn ich meines Bruders Adresse nicht mehr finde, lass ich das auch im netten INTER suchen.

Letzendlich bin ich mir unschlüssig, ob die Informationsflut nun VORTEILE ODER NACHteile hat, auf alle Fälle hat sie seeeeeeeeeeehrr viele Teile.

So überlege ich mir, ob ich nicht besser NICHT mehr in der VD vertreten sein sollte. (Das war eine doppelte Verneinung, die keine solche logisch sein sollte, das für Bayern und NICHT-Bayern). Eigentlich fehlt noch ein „Nicht“ vor dem

[Fortsetzung auf Seite 22](#)

Das Dao Forth

C.H. Ting

Amerikanische Forth-Gesellschaft (FIG USA)
Träger des FORML-1999-Preises für den kürzesten Artikel.

Zusammenfassung:

Michael Ham hat Forth wie folgt beschrieben: „Forth ist wie Dao. Es ist ein Weg, den man wahrnimmt, wenn man ihm folgt. Seine Zerbrechlichkeit ist seine Kraft, seine Einfachheit ist seine Richtung“. Der vorliegende Artikel ist sehr kurz. Er charakterisiert Forth mit all seinen Eigenschaften und versucht dabei, Hams Beschreibung noch weiter zu vereinfachen.

:

Ting

(übersetzt von Fred Behringer)

Forth Interest Group International (FIG USA)

Wollen Sie mit der ganzen Welt verbunden sein und dabei Ihr Englisch perfektionieren ? Amerika ist ein wesentlicher Teil der ganzen Welt. Zumindest, was Forth betrifft. Über tausend Mitglieder aus allen Ländern sind bei uns.

Werden Sie auch ein Mitglied in der Amerikanischen Forth-Gesellschaft (FIG-USA).

Für 45 Dollar im Jahr (Studenten zahlen 18 Dollar) bekommen Sie 6 Hefte unserer Vereinszeitschrift Forth Dimensions und genießen auch sonst verschiedene Vorteile. In den Heften erfahren Sie Forth-Neuigkeiten aus aller Welt, neue Produkte, Literatur, Forth-Ideen, fundiertes Wissen, Artikel auch für Einsteiger, Projekte, Leser-Diskussionen, Quelltexte, Hinweise auf Internet-Verbindungen, kostenlose Forth-Systeme und vieles mehr. (Für Übersee-Porto müssen wir leider noch 15 Dollar hinzurechnen).

Unmittelbare Informationen über uns bekommen Sie, wenn Sie auf der Homepage der Deutschen Forth-Gesellschaft "Links zu anderen Forth-Organisationen" und dann "Forth Interest Group (USA)" anklicken.

Ansonsten bekommen Sie Auskünfte über das amerikanische Forth-Büro:

Forth Interest Group
100 Dolores Street, suite 183
Carmel, California 93923
USA

oder auch vom Redakteur, Marlin Ouverson, unter der E-mail Adresse:

E-Mail: office@forth.org oder
editor@forth.org



REILHOFER KG;

Frühlingsplatz 9

Tel: 08131 92059

Fax 08131 97447

85757 Karlsfeld / München

Möglichkeiten der Schadensfrüherkennung unter dynamischen Bedingungen

Ein Verfahren zur Trennung betriebsbedingter von schadensbedingten Signalen

Um Schäden zu finden ist der Körperschall der richtige Weg. In der Praxis erschweren jedoch die nachfolgend aufgelisteten Zusammenhänge die Schadensfrüherkennung. Allein die Tatsache, daß eine Maschine aus mehreren beweglichen Bauteilen besteht, erzwingt, daß eine funktionierende Lösung nicht nur den eingetübten Sonderfall erkennt, sondern jeden Schadensfall, der vorkommen kann.

Die folgende Liste zeigt, daß die betriebsbedingten und die schadensbedingten Schwingungsemissionen recht ähnlich sein können.

Der nachfolgende Beitrag wurde der VD mit freundlicher Genehmigung der Reilhofer KG zum Abdruck zur Verfügung gestellt und entspricht dem Vortrag von Johannes Reilhofer anlässlich der Jahrestagung 1999 in Oberammergau.

Liebe FORTH-Kollegen,

der nachfolgende Text beschreibt ein Verfahren, mit dem heute alle deutschen Automobilhersteller ihre Getriebe in der Entwicklungsphase prüfen. Das Target-System ist mit dem Harries RTX2000 ausgestattet, also bestens für FORTH geeignet. Auf dem Interface zum Menschen, dem PC, läuft das UR-FORTH von LMI. Ohne FORTH hätten wir uns nie gegen die übermächtige, internationale Konkurrenz durchsetzen können. Mit FORTH waren wir immer schneller als die anderen, ähnlich dem Hasen- Igelwetlauf aus dem Märchen. Was uns an Kapital fehlte, ersetzten wir durch das bessere Werkzeug.

Ich komme zum Thema:

Die Forderung an moderne Getriebe ist: kleiner, leichter, leiser und natürlich billiger, also so abgespeckt, daß es der Kunde gerade noch nicht merkt. Das bedeutet, daß der Grat zwischen - zu aufwendig oder nicht gut genug - noch schmaler geworden ist.

Der Getriebeentwicklung folgt der Prüfstandsversuch. Hier muß es sich erweisen ob Theorie und Praxis zum gleichen Ergebnis führen. Und weil man dem Schicksal nicht traut, prüft man so wirklichkeitsgetreu wie nur möglich, also dynamisch.



- Ein Getriebe liefert auch dann einen komplexen Körperschall, **obwohl es keinen Schaden hat.**
- Dieser Körperschall ändert sich dauernd, auch dann, **wenn sich kein Schaden entwickelt.**
- der schadensbedingte Emissionsanteil ist, wenn er irgendwann auftritt, **nur ein Bruchteil der üblichen betriebsbedingten Körperschallemissionen.**

Diese ungünstigen Voraussetzungen erfordern eine Technik, die es erlaubt, die betriebsbedingten von schadensbedingten Emissionen zu trennen. Ein Weg dazu führt über die unterschiedlichen Verhaltensmuster der beiden Emissionsarten.

Der delta-ANALYSER, der im Kern auf der Fourieranalyse basiert, ist das Werkzeug, mit dem diese Aufgabe bewältigt wird. Im Folgenden wird das speziell Neue an seinem Funktionsablauf erläutert.

Beschreibung der Umgebungsbedingungen

Ein entscheidender Schritt auf der Suche nach einem allgemein anwendbaren Diagnosesystem war die Erkenntnis, daß selbst der beste Getriebeprüfstand nur eingeschränkt in der Lage ist, stabile Lastpunkte zu erzeugen.

Die Folge: Die Körperschallschwingungen, die auf einen Schaden hinweisen würden, werden um ein Mehrfaches an Emissionsschwankungen durch allgemeine Laufinstabilitäten überdeckt. Nur Schäden im fortgeschrittenen Zustand erzeugen soviel Geräusch, daß sie sich merklich vom **betriebsbedingten "Chaos"** abheben und deshalb sichtbar werden. Als Beispiel sei angemerkt, daß eine Drehmomentsschwankung von 1% eine Modulation aller Spektrallinien von ebenfalls etwa 1% zur Folge hat. Pitting auf einem Zahn, mit einer Flächenausdehnung von z.B. 2 mm², geht dagegen in seiner spektralen Auswirkung unter - aber das wäre genau der Scha-



densbeginn, den man sucht.

Auflistung von betriebsbedingten Beeinflussungen des Frequenzspektrums

- Die Zahneingriffs-Emissionen ändern sich mit der Drehzahl und mit dem Drehmoment
- Die Laufgeräusche der Lager sind von Drehzahl und Drehmoment abhängig
- Der Eingriff der Maschine am zu bearbeitenden Material (z.B. Mahl- und Knetprozeß) beeinflusst das Spektrum
- Diese störenden, betriebsbedingten und instabilen Emissionen besitzen immer ein Vielfaches an Energie im Vergleich zu denen, die durch den Schadensbeginn bedingt sind

Dieser betriebsbedingte und eingeschränkt chaotische Zustand läßt sich beschreiben

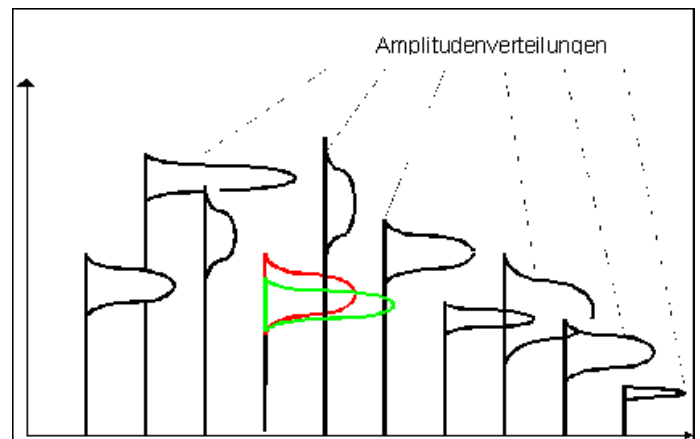
- Das Frequenzspektrum ändert sich zwar dauernd, aber
- es ändert sich stochastisch in einem eingeschränkten Bereich.
- Jede einzelne Spektrallinie zeigt dabei eine individuelle Amplitudenverteilung, (2048 Linien beim delta-ANALYSER)
- Diese Amplitudenverteilung ist typisch für den laufenden Prozeß. Sie ist stabil, solange kein Schaden beginnt
- Sie läßt sich beschreiben und überwachen

Voraussetzungen für die Schadensfrüherkennung an einer komplexen Struktur, wie sie z.B. ein mehrstufiges Getriebe darstellt:

- Der überwachte Frequenzbereich muß so niedrig liegen, daß man mit Sicherheit davon ausgehen kann, daß alle Schadens-

emissionen jeden Ort innerhalb der überwachten Maschine, über die Schallausbreitung, erreichen.

- Die Eigenresonanzen von Wellen, Rädern und Gehäusen dürfen bei den untersuchten Frequenzen noch keine Schwingungsknoten und damit Auslöschungen bilden können.
- Das Überwachungssystem muß sich auf die Drehzahlen der Maschine synchronisieren können. Damit bestimmt der Prüfling die Abtastrate und Drehzahländerungen werden dadurch nicht, fälschlicherweise, als Schadensemissionen interpretiert



Das Überwachungssystem muß das betriebsbedingte "Chaos" in einer Lernphase als Parameterbasis ablegen können.

Das stilisierte Frequenzspektrum zeigt übertriebene Instabilität der Amplituden

Im Beispiel ist bei der vierten Linie eine Veränderung der Amplitudenverteilung angedeutet. Die neue Verteilung ist, verglichen mit der früheren, nicht mehr deckungsgleich. Diese systematische Änderung wäre ein Indiz für einen Schadensbeginn.

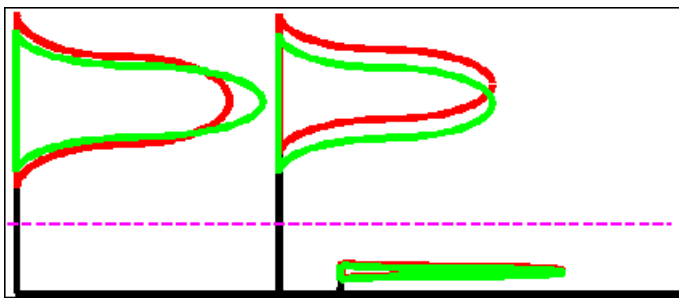
Schadensbedingter Zustand

- Spektrallinien ändern ihre Amplitudenverteilung
- Die Konvergenz zur ursprünglichen Amplitudenverteilung geht verloren
- Es entstehen zusätzlich neue Spektrallinien



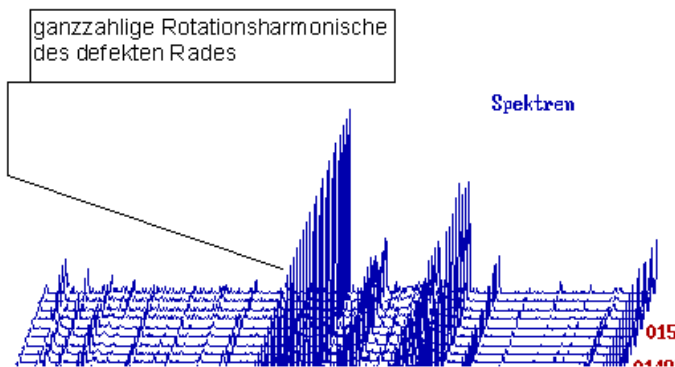
- Die geschädigte Maschine verliert ihre rotationssymmetrischen Eigenschaften. Sie fängt an zu “humpeln” und erzeugt dadurch ganzzahlige Vielfache der Rotationsordnung des geschädigten Bauteils

Das nachfolgende Bild zeigt die Schwierigkeit, mit der man bei der Bestimmung von Toleranzbandbreiten bei den klassischen Analyseverfahren zu kämpfen hatte. Im Beispiel gäben die beiden dominierenden Linien das Toleranzband vor. Die dritte, neu entstandene Linie, die in diesem Beispiel ebenfalls einen Schadensbeginn gezeigt hätte, kommt bei vorgegebenem Toleranzband nicht mehr zur Auswertung.



Beispiel für drei Möglichkeiten einer Schadensankündigung

Das folgende Bild zeigt diesen Zusammenhang an einem



wirklichen Beispiel.

Bei dem Schaden handelt es sich um einen Anriß am Zahnfuß. Das geschädigte Rad diente der Umkehr der Drehrichtung beim Rückwärtsgang eines Pkw-Getriebes.

Selbst in dieser ungenauen Wasserfalldarstellung ist noch eindeutig zu sehen, daß die Amplitudenschwankungen der dominierenden Spektrallinien größer sind, als die, die den Schaden anzeigen. Hätte man an ihnen eine Alarmschwelle orientiert, dann wäre der Zahnschaden bis zum Zahnbruch unsichtbar geblieben.

Die Beispiele zeigen, daß nicht die Größe oder die relative Stabilität einer Spektrallinie ein Schadensmaß sind. Die Scha-

densindikatoren sind abhängig von der Konstruktion des Prüflings und die ist, bezogen auf die Schadensemissionen, eher zufällig.

Um nun die schadensbedingten Schwingungen zu erkennen, lernt der delta-ANALYSER vorher in einer Lernphase das betriebsbedingte Verhalten. Dazu erfaßt er eine vorgebbare Zahl von Signalspektren. Für jede einzelne Spektrallinie findet er das Bildungsgesetz ihrer Amplitudenverteilung. Diese prozeß- und maschinenabhängigen Verteilungen beschreiben die Grenzen für ein Verhalten, das “noch” in Ordnung ist. Ein systematisches “Davondriften” einer oder mehrerer Amplitudenverteilungskurven ist ein Indiz für eine Verhaltensänderung einer, am Gesamtprozeß beteiligten, Komponente.

Das systematische Davondriften überlagert sich dem stochastischen Verhalten der Amplitudenverteilung. Es liefert damit die Information über den Schadensbeginn.

Der entscheidende Vorteil des Verfahrens liegt darin, daß es selbst die Signalschwellen für eine Schadenserkenntnis findet. Und das ohne, daß vorher Schadensart und Schadensort bekannt sein müssen.

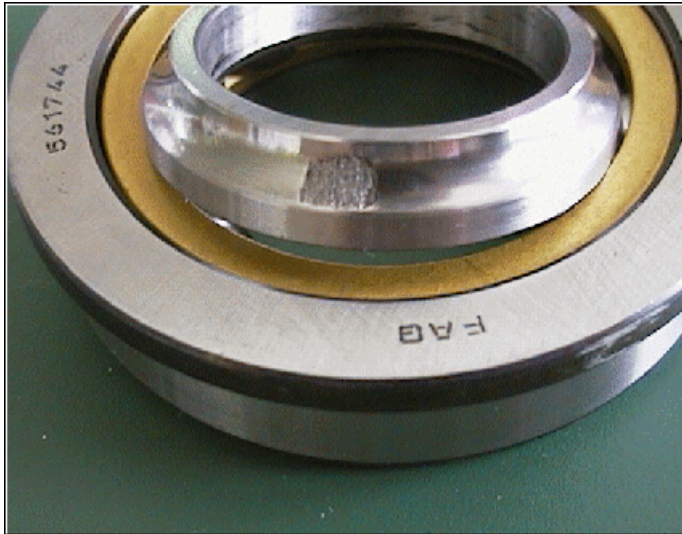
Das **Statistische Verfahren**, das auf der Analyse der Amplitudenverteilungen und der Beobachtung der Maximal- und Minimalwerte basiert, stellt den Fortschritt in der Schadensfrüherkennung dar. Es wurde deshalb zum Patent angemeldet. Das Verfahren erlaubt dem Anwender, ohne Kenntnis der schwingungstechnischen Zusammenhänge, das erfolgreiche Überwachen einer Maschine.

Wenn ein Getriebe möglichst wirklichkeitsgetreu geprüft werden soll, dann wird man es mit einer Betriebslast fahren, die variierende Drehzahlen und Drehmomente aufweist. Das bedeutet, daß sich die Signalspektren noch erheblich stärker ändern, als das bei einem stabilen Lastpunkt der Fall ist. Für diese Zyklenversuche erfaßt der delta-ANALYSER das Spektralkollektiv eines kompletten Blocks und vergleicht ihn mit den folgenden. So ein Block kann beispielsweise beim Test eines Automatikgetriebes das Hochbeschleunigen durch alle Gänge sein, Berg- und Talfahrten und bremsender Schubtrieb. Selbst rasante Beschleunigungsverläufe und abruptes Abbremsen werden in ihren Veränderungen des Spektrums solange toleriert, solange sich kein systematisches Driften überlagert.

Spektrale unter dynamischer Betriebslast entstehen, unterscheiden sich von anderen nur dadurch, daß ihre Amplitudenverteilungen breitere Gaußkurven produzieren. Die Methode der Auswertung bleibt jedoch für alle Lastfälle die gleiche.

Das folgende Bild entstand bei PORSCHE. Die vorzeitigen Spikes im Flächenindex entstanden beim 2-fachen Anhalten des Prüfstandes.

Übrigens – FORTH besucht sein Ursprungsland. Wir haben



CFA2NAME

Dipl.-Ing. Wolfgang Allinger
Brander Weg 6
D-42699 Solingen
Tel 0212 / 66811
eMail: All@business.kbbs.org

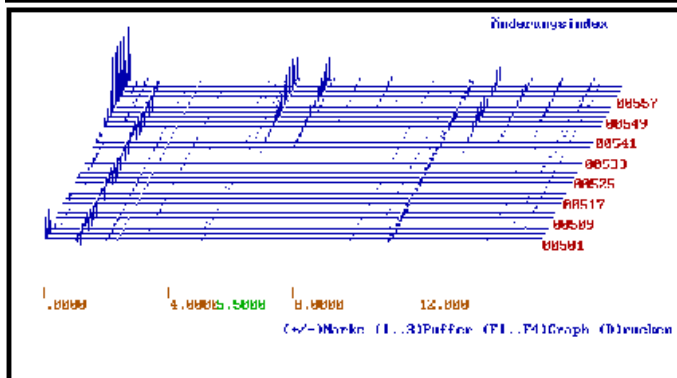
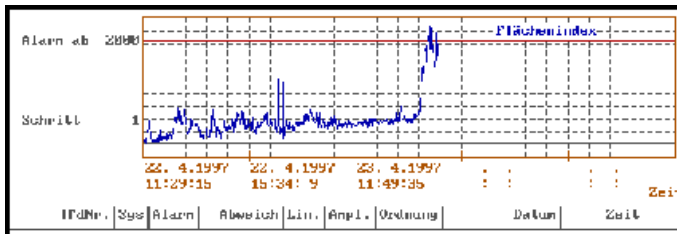
CFA Suchereien
oder
'Wer hat mich da in den A.... gebissen?'

jedenfalls bei General Motors und bei Crysler die ersten 9 Prüfstände mit dieser FORTH-Anwendung ausgerüstet. Auch in den USA waren wir die schnelleren wegen FORTH.

Tps Lauf 1
Type: Prüfling 1
Zähne: 9 / 31
Laufzeit: 12 h, 31 min
bei 6000 1/min

Schaden: Pitting im 4 - Punkt - Lager Antriebswelle
Schadensverlauf und Schadensbild
am delta - Analyser

Johannes Reilhofer



Das Internet läßt uns doch mehr tun, als die Vervielfältigung der VD das kann. Diese Ausgabe der VD wird in ca. 4 Wochen auch über die Homepage der FORTH-EV aufrufbar sein. Dann können Sie die Photos und Skizzen dieses Vortrages in Farbe bestaunen.

red

0. Was soll das?

Ich habe ständig Probleme rauszufinden, wer/was/wie/wo sich daneben benommen hat [na gut, im RL (real live) bin ich's meist selber :-)] aber bei F-PC Entwicklungen, ganz besonders mit Multitasker, kommt es immer wieder vor, daß irgendwas zur Runtime in die Hose geht. Mit viel Glück hat man noch einen Return-Stack, den man untersuchen könnte, um herauszufinden, wo es geknallt hat.

Ach so, ich muß noch erwähnen, daß der von mir modifizierte Multitasker aus dem F-PC inzwischen auch "ABORT" innerhalb einer beliebigen Task kann. Tom Zimmer, Dr. Ting u.a. haben immer behauptet, das ginge nicht. Hihi, alles können die auch nicht :-)

Also bei mir heißt der Multitasker, A für All oder advanced :-)

Wenn man den Return-Stack zu Fuß untersuchen will, kann man sich besser eine Eintrittskarte zu einer Klapsmühle beschaffen, das ist viel gesünder.

Also was mich interessiert, ist eine Liste der Worte, die ihre Return-Adresse auf den RP hinterlassen haben und von wem/wo die aufgerufen wurden. Dies sind die jeweiligen Forth-IP Werte.

Ich hab dazu einige kleine Tools im Multitasker integriert, die mir interaktiv den RP durchforsten.

u.a.

```
: SEE-RP ( RP -- ) \ RP 'disassembler'  
: UNRAVEL ( aRP Rdepth -- ) \ try to analyse the  
\ RP content
```

Weiterhin möchte ich auch manchmal wissen, auf welcher Definition eine Tasks des Multitaskers gerade hängt. Auch da habe ich nur den Forth-IP.

1. Kernproblem

...ist, daß auf dem RP, sowie in den Tasks, immer nur die IP Werte bekannt sind und zwar in der Form Xseg:offset
Aber es gab keinerlei Hilfsmittel, um aus diesem IP Wert auf



den Namen der Definition zu schließen, die zu diesem IP gehört. Ich habe viele vergebliche Anläufe unternommen, um dieses Problem zu lösen.

2. Lösungsansatz

Ein Tip von Tom Zimmer, der im DEBUG auch den Namen der Worte finden mußte, sowie ein Versuch UNRAVEL den RP zu analysieren von Ulli Hoffmann, brachten mich auf die richtige Spur. Auch Arndt Klingelberg hat mir geholfen.

Im Xseg sind die IPs abgelegt, die der Reihe nach per NEST abgegrast werden. Bei Xseg:0000 fängt so eine Liste immer an, ich brauche nur [:-(] das Wort zu finden, welches fragliches Xseg in seiner Definition hat.

Es gibt 2 (?) Definitionen, die sowas machen:

docolon für :-definitionen und dodoes für CREATE DOES>

Also muß man im Code-Segment =Cseg nach docolon's bzw. dodoes's suchen, deren Body auf das fragliche Xseg zeigen. Wenn ich den Body kenne, komme ich über >NAME usw. einfach ans Ziel.

3. Ausführung

Im xseg2cfa.seq durchsuche ich in seg>cfa das codeSEG:0100 bis zum DP ab.

:-definitionen müssen einen Sprung (jmp) nach docol haben, DOES> definitionen müssen einen Aufruf (call) nach dodoes haben.

D.h. ich suche zuerst nach einer :-definition, wenn das nicht gefunden wurde, suche ich nochmal nach DOES> definitionen.

Ein jmp hat den Maschinencode \$E9 und ein Call den code \$E8, die Maschinencodesequenzen sind 3 Byte lang.

Wenn eine \$E9docol bzw \$E8dodoes gefunden wurde, wird nachgesehen, ob im Body (die nächsten 2 bytes) der fragliche Wert von Xseg drinsteht, ist das der Fall, so haben wir den Täter. Andernfalls müssen wir 5 Byte weiter erneut suchen.

Man muß nur noch wissen, daß im F-PC relative Xseg Werte drinstehen, der Anfang steht im Value XSEG.

4. weitere praktische Hilfsmittel

```
: SEE-RP ( RP -- )          \ RP 'disassembler'
```

Da im RP nicht nur IP Werte stehen, kann man leider keine vollautomatische RP Analyse machen. Mit SEE-RP kann man aber verschiedene Arten der RP Darstellung wählen und/oder sich im RP bewegen.

```
: UNRAVEL ( aRP Rdepth -- ) \ try to analyse the
                             \ RP content
```

Unravel (engl. für abwickeln...) stammt ursprünglich von Ulli Hoffmann und interpretiert ab einer addr. im RP für eine vorgegebene Tiefe alle Einträge als Xseg Werte. Ich zeige nun auch die evtl. Namen dazu.

Unravel wird in der neuen (?ERROR)' benutzt, dies wird bei einem ABORT durchlaufen.

5. Codefragment UNRAVEL und (?ERROR)' als Beispiel (aus multaskA.seq)

```
: UNRAVEL          ( aRP Rdepth -- )
\ ALL980602+
\ try to analyse the RP content

    ." RP: "
    SWAP 2 + SWAP   \ adjust for 1st Xseg
    1 MAX 2 cells * BOUNDS DO
    I @ DUP ." x" H. ( -- seg ) X.ID SPACE
    4 +LOOP

\ ALL980602-
;

\ ----
2variable %%addrlen

\ -----

: (?ERROR)'       ( adr len f -- )
  IF
    %%addrlen 2!   \ *** save addr len
    PLOP           \ *** an interesting noise
    MULTIabort    \ *** let UP .2 TCB0, noops
                  \ PAUSE ..., SINGLE
    \ *** DOUSER-VARIABLE DOUSER-DEFER
    \ know about USER !!!
    DEFAULT-SET   \ housekeeping chain
    ERRFIX
    SP0 @ SP!     \ reset parameter stack
                  \ as of TCB0 !!!
    %%addrlen 2@  \ *** retrieve addr len

  2drop
  $buf count     \ drop and assume $buf as
                  \ buffer works \ ALL980404
  ABS 79 MIN     \ limit to 79 chars
                  SLOW
                  done   \ setup a real CRLF ...

\ ALL980oct05
  3 SPACES
\ akg93feb18
  TYPEW SPACE   \ type a DOS-type with line wrap
\ akg93feb19

\ try to analyse the RP content:
\ ALL980602
  RP@ 7 UNRAVEL \ good enough for me
\ ALL980602

  QUIT         \ QUIT should be patched to QUITnew
               \ while loading new-QUIT.seq
  ELSE
    2DROP
  THEN ;

' (?ERROR)' IS ?ERROR

\ #####
```

TCB0 ist der 'Task Control Block' für die Task0. In meinem erweiterten Multitasker ist das die Operator Task. Sie ist identisch mit dem Block von USER Variablen, die auch ohne den Multitasker benutzt werden.



Ich hab immer noch Probleme, daß manchmal eine völlig geschredderte addr an ?ERROR übergeben wird, in \$buf steht aber immer noch der String, der zu dem fraglichen ABORT gehört, also schmeiße ich die Textaddr. weg und benutze den Inhalt von \$buf. Arndt hat ähnliche Probleme beobachtet, aber auch noch keine Lösung gefunden.

6. Codefragment SEE-RP als Beispiel (aus multaskA.seq)

```

: .RP ( aRP -- aRP' ) DUP
  DUP H.$ \ aRP
  2@ 2DUP .seg:off \ IPreturn
  DUP -1 = IF
    2DROP
    ." can't fetch $xxxx:FFFF ! " BEEP
  ELSE
    OVER ." : " X.ID ." ... "
    @L >NAME .ID ." ... ; "
  THEN
    2 CELLS + ( aRP -- aRP' )
;

: SEE-RP ( RP -- )
  1 ?depth
  CR ." <ESC> terminates, 'U' unloops, '+û -û'
  RP, 'R' dumpRP, 'S' .SH "
  CR ." $1234 $1234:5678 :-def ...
  CR ." RP: ES:IP :-def ...
  nameES:IP ...
  BEGIN
  CR .RP ( RP -- RP+2cells )
  KEY CASE
    27 OF ." <ESC>" DROP EXIT ENDOF
    '+' OF ." +û" 1 CELLS - ENDOF
    '-' OF ." -û" 3 CELLS - ENDOF
    $D0 OF ." +û" 1 CELLS - ENDOF
    $C8 OF ." -û" 3 CELLS - ENDOF
  UPC
    'U' OF ." unLOOPed" 6 + ENDOF
    'S' OF 2 CELLS - ." show SP"
    CR .SH ENDOF
    'R' OF 2 CELLS - ." show RP... "
    DUP 16 DUMP ENDOF
  OTHERCASE
  ENDCASE
  AGAIN ;
  -----

```

Die '+-' keys werden gebraucht, um den RP wieder zu 'synchronisieren', wenn mal keine IPs draufstehen, bzw. 'U' um DO..LOOP Einträge zu überspringen.

7. Schluss

...für heute, falls Rückfragen: anrufen oder mailen :-)

8. Anhang

```

\\ xseg2cfa convert a absolute Xseg-paragraph to
the cfa \ ALL980610

this is very useful for chasing bugs in MULTItasker,
where you usually only know the forth-IP, i.e. a
Xseg:offset to X-space.

\ ALL98jul09 (X.id) mod for FPC2000
ALL980610 v0.02 improved comments
ALL980529 v0.01 added X.ID ...
ALL980527 v0.00 idea stolen from DEBUG.seq
according to an eMail hint from
Tom Zimmer

```

```

{
  AUTOTITLE
  \ -----
  defined seg>cfa nip 0= #IF
  \if 'docol ' HEX @REL>ABS
  \ CONSTANT 'DOCOL
  \if 'dodoes ' FORTH @REL>ABS @REL>ABS
  \ CONSTANT 'DODOES
}
: advance ( a1 u1 u2 -- a+u2 u1-u2 | u1<u2: a+u1 0 )
  /string exit
  2DUP U< IF UMIN DUP THEN
  ROT OVER + -ROT - ( -- a+u2 u1-u2 )
;
: find_ ( a1 n1 -- a2 n2 ) \ find any :-definition
  begin $E9 ( jmp ) scan ( -- a$E9 n2 | aZ 0 )
  over @rel>abs 'docol <> over and
  while 3 /string ( -- a+3 n-3 )
  repeat ;
: find_dodoes ( a1 n1 -- a2 n2 )
  \ find any DOES>-definition
  begin $E8 ( call ) scan ( -- a$E8 n2 | aZ 0 )
  over @rel>abs @rel>abs 'dodoes <> over and
  while 3 /string ( -- a+3 n-3 )
  repeat ;
: seg>cfa ( seg -- cfa f1 )
  \ find cfa given the physical segment
  \ t=found, f=not found
  xseg @ - >r \ convert to XSEGrel
  \ search for :-def w/ (body)=XSEGrel
  $100 here $100 -
  \ dictionary=codeSEG: $100 (DP-$100)times
  begin find_ ( -- a$E9docol n | a' 0 )
  over >body @ r@ <> over and \ (body)=Xrel?
  while 5 /string
  \ no, advance behind 5byte $E9....
  repeat ( -- a n )
  dup 0=
  if 2drop
  \ search for DOES>-part w/ (body)=XSEGrel
  $100 here $100 -
  begin find_dodoes over
  @rel>abs
  >body @ r@ <> over and
  \ (body)=Xrel?
  while 5 /string
  \ no, advance behind 5byte $E8...
  repeat
  then
  r>drop
;
#THEN
fpcver#ak 4232. d> #IF \ t= NUVOLO9807 , f= 4.17
: (X.id) ( Xpara -- )
  \ show the name where XPARAGRAPH .2
  SEG>CFA ( xseg -- cfa ? )
  DROP >NAME (name) type \ no spaces
\ ALL98jul09
;
#ELSE
: (X.id) ( Xpara -- )
  \ show the name where XPARAGRAPH .2
  SEG>CFA ( xseg -- cfa ? ) DROP >NAME (%.ID)
  \ no spaces
;
#THEN
: X.ID ( Xpara -- ) (X.id) SPACE ;
\
: T @> SEG>CFA +XSEG SPACE X.ID ;
\ should type 'SEG>CFA'
\ end of ALL's garbage

```



Hashing

- Teil 2 -

Friederich Prinz
 Hombergerstraße 335
 47443 Moers
 Tel.: 02841 / 58398 (Q)
 E-Mail: Friederich.Prinz@t-online.de

Im ersten Teil dieses Aufsatzes (VD 04/98) habe ich beschrieben, wie die Fragestellung nach dem Hashing an mich herangetragen wurde, was das Hashing per Definition ist (Duden Informatik) und wie Tom Zimmer das Hashing im ZF realisiert hat. Der zweite Teil wird zeigen, wie sich bei konkreten Versuchen ein Hashing zu implementieren, ein Forthsystem regelrecht 'entwickelt'. Die in diesem Teil beschriebenen Erfahrungen mit ZF lassen sich ohne größere Abstriche auf F-PC übertragen.

Internes zu Vokabularen

Der Johannes möchte „realen Code“ sehen. Ich möchte versuchen, ihm welchen zu zeigen - und stelle fest, daß die Arbeiten „drum herum auch nicht ohne“ sind. Zum Beispiel habe ich mich bisher nie dafür interessiert, wie ZF intern seine Vokabulare verwaltet. Wie in den vorhergegangenen Worten aber zu sehen ist, sind die Vokabulare maßgeblich an der Erzeugung der Threads beteiligt. Das mußte ich mir erst einmal ansehen. Was es dabei zu „entdecken“ gab, fasse ich hier zusammen. Ich versuche, meine „Entdeckungen“ ausführlich zu beschreiben. Für die Cracks unter den Lesern gerät das vielleicht etwas zu ausführlich. Ich mach's halt für die etwas weniger versierten Leser, die von „ihren“ Systemen noch nicht wissen „wie das funktioniert“.

An dieser Stelle muß ich auf einen Kurs hinweisen, der im Frühjahr 1992 in der Moerser Forthgruppe angeboten wurde: " FORTH intern - Innere Strukturen in ZF". Daß ich auf einen mehrere Jahre alten Aufsatz/Kursteil von mir verweise, hat weniger mit meinem (unbestreitbar nicht unterentwickelten) Ego zu tun, als vielmehr damit, daß in diesem Aufsatz die inneren Strukturen in ZF's Definitionen beschrieben sind. Darin werden nicht nur Begriffe wie CFA, PFA, LinkFelder etc. erklärt, sondern auch die sich zum Beispiel bei Schleifen innerhalb der Worte aufbauenden Strukturen. Wer in den letzten zwei Jahren das ZF von mir bekommen hat, verfügt auch über diesen Text - und kann dort noch einmal etwas detaillierter nachsehen, was ich hier zu beschreiben versuche...

Die meisten (alle ?) FORTH-Systeme haben ihre Wörterbücher in mehrere Vokabulare aufgeteilt. Die Namen der einzel-

nen Vokabulare sind in den Wörterbüchern ebenso Namen, wie die Namen aller anderen, „gewöhnlichen“ Definitionen auch. Zum Beispiel steht im ZF der Name des Vokabulars DOS zwischen den Worten DOSVER und HDOS1. Dem Namensseintrag ist nicht anzusehen, daß es sich bei ihm um keine „Definition im klassischen Sinne“ handelt.

Über die CFA (CodeFeldAdressen) der Vokabulare werden aber, wie zuvor schon zu sehen war, die „Einsprungadressen“ der Threads in die Hashingtabelle berechnet. Um herauszufinden, welches Wort im Dictionary nun ein Vokabular beschreibt, und welches nicht, ließe sich zu jedem Eintrag ein Flag denken, eine Nachricht gewissermaßen, die einen Namen im Wörterbuch als Vokabularnamen kennzeichnet.

Will man sich eine Liste aller Vokabulare ansehen, oder will man einfach „auf die Schnelle“ alle CFA aller vorhandenen Vokabulare auf den TOS bekommen, muß der Interpreter nun alle FORTH-Worte auf eine solche Kennzeichnung überprüfen. Geschickter läßt sich eine solche Abfrage auf Vokabulareinträge lösen, wenn diese von vorneherein anders organisiert werden. Tom Zimmer fädelt z.B. im ZF alle Informationen zu Vokabularen auf eine separate „Kette“, was recht schnell deutlich wird, wenn man sich das Wort ansieht, mit dessen Hilfe sich die Namen aller Vokabulare auf den Bildschirm ausgeben lassen...

```

: VOCS ( -- ) \ gibt die Namen aller Vokabulare auf
               \ den Bildschirm aus )
CR
VOC-LINK @    \ ...hier startet die 'Kette' aus
               \ Adressen...
BEGIN
DUP          \ Die Adressen werden in der
               \ Kette mit dem jeweils
#THREADS 2*  \ doppelten Abstand der maximalen
               \ 'Fadenzahl'
-            \ gespeichert (siehe Skizze)
BODY>       \ ...von der PFA zur CFA...
>NAME       \ ...von der CFA zur NFA...
.ID         \ Namen aus dem Headersegment
               \ auslesen
@           \ nächsten 'Knoten' auf der Kette
               \ bearbeiten,
DUP 0= UNTIL \ bis eine '00' anzeigt,
               \ daß alle Einträge
DROP ;      \ bearbeitet wurden.
    
```

Die Variable VOC-LINK enthält also eine Adresse, an der eine Kette aus Adressen beginnt. Wir können uns die in der Kette enthaltenen Adressen manuell auf den TOS rufen:

```

VOC-LINK @      --> 17580 (44AC) - 32
                  = PFA von      BUG
VOC-LINK @ @    --> 14418 (3852) - 32
                  = PFA von      HIDDEN
VOC-LINK @ @ @  --> 13879 (3637) - 32
                  = PFA von      ROOT
VOC-LINK @ @ @ @ --> 12792 (31F8) - 32
                  = PFA von      USER
VOC-LINK @ @ @ @ @ --> 7804 (1E7C) - 32
                  = PFA von ASSEMBLER
VOC-LINK @ @ @ @ @ @ --> 4954 (135A) - 32
                  = PFA von      DOS
VOC-LINK @ @ @ @ @ @ @ --> 303 (012F) - 32
                  = PFA von      FORTH
VOC-LINK @ @ @ @ @ @ @ @ --> 0
    
```



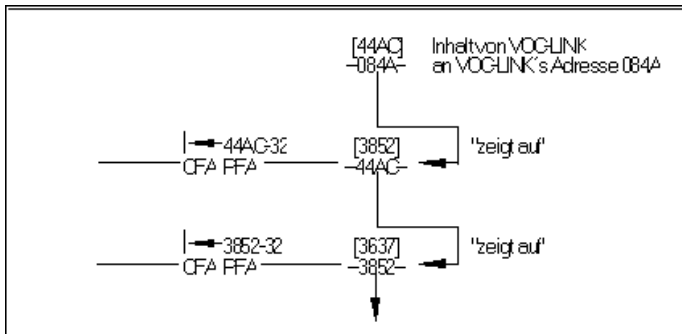
Worte am (seidenen) Faden

Die Angaben dieser Liste lassen sich leicht überprüfen:

12792 32 - BODY> >NAME .ID USER ok

(Die Anweisung '32 -' entspricht hierbei '#THREADS 2* -'.)

Hier ein Versuch, die Verkettung sichtbar zu machen:



Der Inhalt der Variablen VOC-LINK ist die Adresse einer Speicherstelle. Der Inhalt dieser Speicherstelle ist die Adresse einer Speicherstelle. Der Inhalt ... usw.

Das Ganze wird fortgesetzt, bis eine der so verketteten Speicherstellen den Inhalt '00' hat. Dort ist dann die Kette zu Ende. Wenn man von einer dieser Speicherstellen 32 Byte zurück geht, gelangt man an eine Speicherstelle, die die PFA (Parameterfeldadresse) eines Vokabularnamens enthält. Noch zwei Byte weiter „zurück“, ist der Speicherinhalt die sogenannte CFA. Von der CFA aus werden die „Einsprünge“ in das Headersegment des Wörterbuches organisiert!

Die „Arbeit am HASHING“ führt immer tiefer in die Strukturen des Systems. Ich finde es spannend zu sehen, wie sich das ZF im wahrsten Sinne des Wortes regelrecht „entwickelt“.

Ich kann allen FORTHern nur empfehlen nachzusehen, wie sich die hier für das ZF beschriebenen Sachverhalte im jeweils benutzten System konkret darstellen. Das schärft die Sinne für das, was „machbar“ ist...

Das ZF hat die Inhalte seines Wörterbuches in mehreren Strukturen organisiert. Eine dieser Strukturen, die ich kurz im Zusammenhang mit der Variablen VOC-LINK beschrieben habe, sind die Vokabulare. Eine andere Struktur sind die bereits beschriebenen THREADS, die Fäden, an denen Worte mit gleichem Hash-Funktionswert in einer linearen Liste aufgereiht werden. Wie aus der Beschreibung von HASH zu sehen war, existieren zu jedem Vokabular maximal 16 solcher Fäden. Die Startadressen der Fäden, die nichts anderes sind, als Offsets auf das Headersegment, sind zum einen abhängig von dem „Hash“ des jeweiligen Vokabulars, und zum anderen bestimmt durch den Anfangsbuchstaben des jeweils gesuchten Wortes.

Die einzelnen Worte, sofern sie sich auf dem gleichen Thread befinden, sind ebenfalls miteinander verknüpft - über das sogenannte Linkfeld. Die LinkfeldAdresse (LFA) ist ein fester Bestandteil der Wortheadere in den meisten FORTH-Systemen, zumindest solange sie auf F83 basieren.

Einsteiger werden sich ein wenig schwer tun, die Links (Verbindungen) in den Headern von ZF oder F-PC zu verfolgen. Diese beiden Systeme halten Wortheadere und -bodies voneinander getrennt. Die Wortheadere liegen in einem anderen Speichersegment als die Bodies. Die Bodies, die im aktuellen Codesegment liegen (das gleich dem Datenssegment ist), lassen sich mit dem Wort DUMP ansehen. Adressen in den Bodies sind mit @ und C@ adressierbar. Die Inhalte der Wortheadere können dagegen weder mit DUMP angesehen, noch mit @ oder C@ ausgelesen werden. Dafür stellt ZF aber die Worte Y@ und YC@ bereit. Mit deren Hilfe läßt sich ein „YDUMP“ selbst erstellen. Das „Y“ steht hierbei für Y-Segment, also für das Segment, in das ZF die Header seiner Definitionen ablegt.

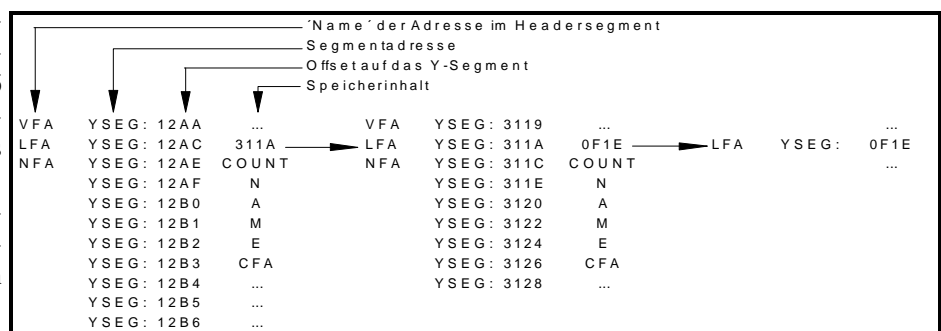
In den Wortheadern (oder sagen wir's auf Deutsch - in den Wortköpfen) sind neben den Namen die NFA, LFA und VFA untergebracht.

NFA = Namensfeldadresse - Adresse, ab welcher der Namenstext einer Definition beginnt. Das erste Byte enthält den Count der Zeichenkette. Von diesem Byte sind allerdings nur die Bits 0-4 für den Count nutzbar, weshalb Definitionsnamen auf 31 Zeichen beschränkt sind.

LFA = Linkfeldadresse - Adresse, in welcher die Adresse des jeweils nächsten Wortes im aktuellen Thread enthalten ist. Über die LFA sind alle Worte miteinander verbunden, die im aktuellen Vokabular den gleichen Hash-Funktionswert haben.

VFA = Viewfeldadresse - Adresse, die nicht zum Standard gehört und meines Wissens nur in ZF und in F-PC implementiert ist. Unter Zuhilfenahme der Variablen FILELIST, die den Start einer Kette aus allen Dateinamen enthält, die zum aktuellen Umfang des Systems herangezogen wurden, kann der Interpretierer beim Aufruf des Wortes VIEW den Quelltext einer beliebigen Definition auf den Bildschirm ausgeben. Für die Überlegungen zum HASHING ist das Viewfeld irrelevant.

Die Adressen NFA und LFA liegen aber im Y-Segment und sind Offsets auf den Anfang dieses Segmentes. Die Verbindung (LINK) zwischen den einzelnen Worten läßt sich so skizzieren:

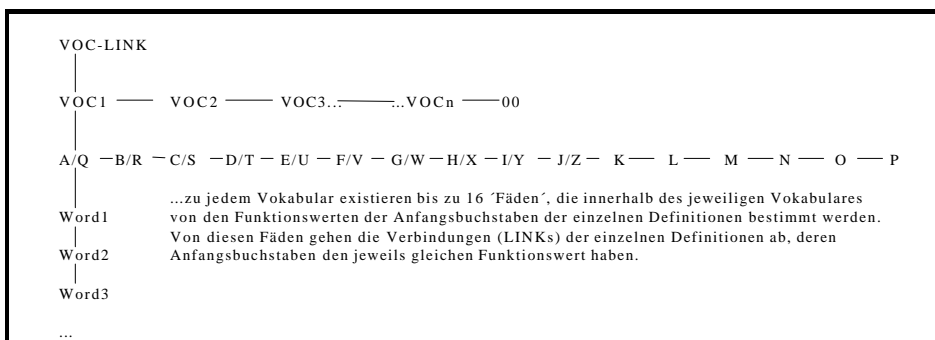




Die LFA enthält die Adresse der nächsten LFA. Die nächste LFA enthält wieder die Adresse der nächsten LFA, usw.. Am Ende der Kette ist der Inhalt der LFA „00“. Das zeigt das Ende der Kette an. Innerhalb des Y-Segmentes sind die Abstände zu den jeweiligen Nachbaradressen (VFA, NFA) immer gleich, 2 Byte plus, bzw. minus auf die Adresse LFA. Die Ablage der Header ist also im Grunde recht einfach organisiert. Interessant ist auch noch, daß am Ende eines Namenseintrages die CFA eingetragen ist. Die CodeFeldAdresse ist die Adresse im CodeSegment, an welcher der Body, der Code eines Wortes beginnt.

Ordnungsliebende Menschen könnten auf den Gedanken kommen, daß es viel angenehmer wäre, wenn die CFA nicht am Ende eines Wortes gespeichert würde, sondern einen festen Platz hätte, wie auch NFA, LFA und VFA. Aber hierzu muß man sich vor Augen halten, zu welchem Zeitpunkt die CFA benötigt wird. Wenn der Compiler die CFA einer Definition in die Parameterliste einer neuen Definition eintragen soll, dann muß er erst einmal im Wörterbuch nachsehen, ob diese Definition überhaupt existiert. Er muß also, nach dem bereits für HASH beschriebenen Verfahren, das Wörterbuch durchsuchen und am Ende mindestens einen Stringvergleich durchführen. NACH diesem Stringvergleich, sofern dieser Übereinstimmung ergeben hat und die angegebene Definition als „gefunden“ betrachtet werden kann, steht der „Zeiger“ auf dem Namenseintrag am Ende des Namens, zeigt also auf die CFA - die jetzt nur noch, ohne weitere Berechnungen und Klimmzüge, ausgelesen werden muß.

Demnach kommen bei der „Durchforstung“ des Dictionarys drei Strukturen zum Tragen, die sich wie folgt skizzieren lassen:



ACHTUNG: Es kann durchaus vorkommen, daß ein einzelnes Vokabular einen oder mehrere Threads NICHT besitzt ! Zum Beispiel existieren im Vokabular BUG keine Worte, die mit dem Buchstaben „A“ beginnen. Darum gibt es dort keinen Thread für einen Funktionswert zu diesem Buchstaben UND diesem Vokabular !

Das ist doch eigentlich alles recht einfach - wenn man die Strukturen einmal kennt. Es bleibt die Frage, wie sich das nachprüfen läßt. Ich habe mir dazu für das ZF ein „WORDS“ einfallen lassen, das einfach alle Worte in allen Vokabularen unformatiert untereinander ausgibt und dabei die Anzahl der

gefundenen Worte mitzählt. Mit „WORDS *.*“ kann ich anschließend prüfen, ob meine Definition etwas ausgelassen hat. Mein Words, das alle Vokabulare, also alle „Links“ abarbeiten soll, heißt „VoclinksAbarbeiten“.

```

\ VARIABLE Totalwords \ ist in WORDS.SEQ definiert
\ \ und zählt die Anzahl der
\ \ gefundenen, bzw. aus-
\ \ gegebenen Worte

: Thread.Abarbeiten ( Thread -- ) \ 'Thread' ist das
\ \ LinkFeld der
BEGIN \ jüngsten Definition
DUP \ LFA duplizieren
L>NAME \ zur NFA
CR .ID TOTALWORDS INCR \ Namen ausgeben
Y@ \ nächste LFA der Wortkette holen
DUP \ wenn die ungleich NULL ist, dann
0= UNTIL \ existiert ein weiteres Wort in der
DROP ; \ Kette...

: Alphabet.Abarbeiten ( Thread -- ) \ Jedes Vokabular
\ \ kann max.16
ASCII A \ Threads haben. Der erste 'Faden'
#THREADS 0 \ setzt auf den Buchstaben 'A' auf...
DO DUP
PAD 1+ C! \ Buchstaben als String nach PAD
PAD 2 PICK HASH @ \ Thread aufnehmen
\ \ und 'hashen'
DUP 0<> IF Thread.Abarbeiten \ wenn der
\ \ buchstabenabhängige
ELSE DROP \ Faden existiert,
\ \ dann abarbeiten,
THEN \ sonst fallenlassen
1+ \ nächsten Buchstaben entspr. dem
LOOP \ nächsten 'Faden'...
DROP DROP ;

: VocLink.Abarbeiten ( -- ) \ Alle im System
\ \ vorhandenen
0 TOTALWORDS ! \ Vokabulare abarbeiten...
1 PAD C! \ Im PAD steht ein String mit
\ \ COUNT 1
VOC-LINK @ \ Start der 'Vokabularkette'
BEGIN
\ \ ..vom Linkfeld der
\ \ Vokabulare
DUP #THREADS 2 * -
\ \ zu deren Parameterfeld-
\ \ adressen...
Alphabet.Abarbeiten
\ \ Die PFA's sind die
\ \ 'Threads'
@
\ \ nächstes Vokabular
\ \ aufnehmen
DUP 0=
\ \ wenn der Zeiger '0'
\ \ enthält, dann
UNTIL
\ \ ist die Vokabularkette abgearbeitet
DROP ;

```

Diese Definitionen will ich, in einer modifizierten Form, dazu nutzen, mit einem „eigenen“ Hashing zu experimentieren, und ZF's Dictionary vollständig „richtig zu hashen“. Dazu sind vorher allerdings noch einige Versuche notwendig, die zeigen sollen, ob die verschiedenen, möglichen Alternativen akzeptable „Kollisionsraten“ aufweisen.

Die bisher detailliert erklärten Worte werden aus Platzgründen ohne Kommentare und in einer kompakten Form einfach nur noch benutzt. Lediglich neue Definitionen und/oder Änderungen beschreibe ich neu.



Worte am (seidenen) Faden

Ich will zum Beispiel wissen, ob meine bisherigen Überlegungen stimmen. Dann müßte es problemlos und ohne *Verrenkungen* möglich sein, alle Worte aus allen Vokabularen in einen separaten Puffer zu kopieren.

```

0 CONSTANT Puffer          \ Offset auf HERE 500 +
CREATE  Tabelle 5000 ALLOT \ für jeden
      \ Funktionswert 1 Byte
      Tabelle 5000 ERASE \ ich will zählen,
      \ wie oft Funktionswerte mehrfach vorkommen...
CREATE  Nenner 20 ALLOT \ hier wird nachge-
      Nenner 20 ERASE \ zählt, wie viele
      \ 'Mehrfachnennungen' vorkommen
0 CONSTANT Flag          \ steuert die Aktionen...
VARIABLE Zaehler        \ Anzahl der kopierten Worte
VARIABLE Laenge         \ Summe aller Counts
-----
: Voc>Puffer ( -- )
  DUP YC@ 31 AND 1+      \ count +1
  DUP >R                \ Alle Worte aller Vokabulare
  0 DO DUP I +          \ samt deren Counts in einen
      YC@ 127 AND      \ Puffer kopieren, der ab
      Puffer I + C!    \ HERE 500 + beginnt.
  LOOP
  DROP
  Puffer C@ 31 AND Puffer C! \ der COUNT wird
  R> Puffer + =: Puffer ; \ sicherheitshalber
      \ noch einmal mit 31 AND verglichen

```

Um die Größe des anzulegenden HashSpeichers beurteilen zu können, sollte man wissen, wie „lang“ ein Wort im arithmetischen Mittel ist. Das Wort `?WORTLAENGE` hilft mir hierbei. Es kontrolliert zunächst die Anzahl der in den Puffer kopierten Worte, die identisch sein muß mit der Anzahl der von `WORDS **` gemeldeten Worte. Gleichzeitig addiert `?WORTLAENGE` die `COUNTS` aller Worte. Eine einfache Division der „CountSumme“ durch die Wortanzahl ergibt die mittlere Länge. Im ZF sind dies, wie sich zeigen wird, fünf Zeichen, plus dem Count. Die mittlere Wortlänge beträgt also 6 Byte...

?Mehrfachnennungen kontrolliert, wie oft ein Wort mehrfach in einem oder in allen Vokabularen vorkommt. Die Mehrfachnennungen werden beim geplanten Hashing noch Probleme bereiten, aber zunächst möchte ich gerne wissen, wie relevant dieses Problem ist. Die hier ausgezählten Mehrfachnennungen beziehen sich auf gleiche `HASH`-Funktionswerte. Allerdings existieren einige Worte in `FORTH's` Vokabularen tatsächlich auch mehrfach...Nun, ich schau mir das erst einmal an...

```

: Mhash ( NFA -- )
  DUP 1+ YC@ 31 AND SWAP \ 1. Zeichen auslesen
  DUP YC@ 1+ 1 DO 1+ DUP YC@\ von 1 bis COUNT+1
      \ nächstes Zeichen
      31 AND \ auslesen und 'Zahl'
      \ draus machen
      I * \ Zahl mit Index
      \ multiplizieren
      ROT + SWAP \ und auf '1.Zeichen'
      \ addieren
  LOOP
  DROP \ Adresse wird nicht mehr gebraucht
  5000 MOD \ 'Quersumme' durch Tabellengröße

```

```

Tabelle +          \ Funktionswert adressiert
              \ Tabelle,
DUP C@ 1+ SWAP C! ; \ das Byte der Tabelle wird
              \ um '1' erhöht...

```

```

: Thread.Abarbeiten ( Thread -- )
  BEGIN
  DUP L>NAME
  -----
  Flag 0 = IF Voc>Puffer THEN
  Flag 1 = IF Mhash THEN
  -----
  Y@ DUP
  0= UNTIL
  DROP ;

: Alphabet.Abarbeiten ( Thread -- )
  ASCII A #THREADS 0
  DO DUP PAD 1+ C! PAD 2 PICK HASH @
  DUP 0<> IF Thread.Abarbeiten
      ELSE DROP
      THEN
  1+
  LOOP DROP DROP ;

: VocLink.Abarbeiten ( -- )
  1 PAD C! VOC-LINK @
  BEGIN
  DUP #THREADS 2 * -
  Alphabet.Abarbeiten
  @ DUP
  0= UNTIL DROP ;

: ?WortLaenge( -- ) \ Kopierarbeit überprüfen und die
      \ mittlere Wortlänge errechnen
  0 Zaehler ! 0 Laenge ! HERE 500 +
  BEGIN
  DUP COUNT + SWAP \ 1. Count + Startadresse ist
  C@ DUP DUP \ COUNT 3-fach auf dem Stack
  0<> IF Zaehler INCR \ COUNT <> 0 , dann Wort
      \ zählen
  THEN
  Laenge +! \ Count aufaddieren zur Countsumme
  0= UNTIL \ ...arbeiten bis Puffer '00' ist
  DROP
  CR ." Worte im Puffer : " Zaehler @ 5 .R
  CR ." mittl. Wortlänge : " Laenge @ Zaehler @ /
  1+ 5 .R
  CR ;

: ?Mehrfachnennungen ( -- )
  5000 0 DO Tabelle I + C@ 2* \ Alle 5000 'Plätze'
      \ in der Tabelle
      Nenner + \ abarbeiten und die
      \ Anzahl der
      DUP @ 1+ SWAP ! \ Nennungen aus-
      \ zählen
  LOOP
  CR ." leere Hashadressen : " Nenner
  @ DUP 5 .R
  10 1 DO CR I ." -fache Nennungen : " Nenner I 2*
      + @ DUP 5 .R
      I * SPACE 42 EMIT SPACE I . 61 EMIT
      DUP 5 .R
  LOOP
  CR 22 SPACES ." -----"
  CR 32 SPACES + + + + + + + 5 .R
  DROP ;

```

Das Ergebnis, „aufgenommen“ mit dem in der VD 4/1993, S.22-24 beschriebenen „Recorder“, wird auf der folgenden Seite dargestellt.

Es gefällt mir noch gar nicht. Trotz der doppelten Namensgebung einzelner Worte in den verschiedenen Vokabularen, verbleiben noch 291 Fälle, in denen der Hashfunktionswert



```
---[ ZF-Forth LOG-File ]----(geöffnet )-----21.05.1994--10:48:25---
```

hashtest

...statistische Infos zum Dictionary...

```
Worte im Puffer      : 1241
mittl. Wortlänge    : 6

leere Hashadressen  : 3910
1 -fache Nennungen : 950 * 1 = 950
2 -fache Nennungen : 129 * 2 = 258
3 -fache Nennungen : 11 * 3 = 33
4 -fache Nennungen : 0 * 4 = 0
5 -fache Nennungen : 0 * 5 = 0
6 -fache Nennungen : 0 * 6 = 0
7 -fache Nennungen : 0 * 7 = 0
8 -fache Nennungen : 0 * 8 = 0
9 -fache Nennungen : 0 * 9 = 0

-----
1241
```

Im Win32FOR ist der „Recorder“ als Werkzeug längst eine Selbstverständlichkeit geworden. Für das ZF mußte ich dieses Werkzeug noch selbst erstellen. Bei Interesse am Quelltext dieses Tools schreiben Sie mir bitte per E-Mail. Ich sende Ihnen die Quelle zum Recorder dann umgehend zu.

fep

```
STOP  DEFINITIONS  FORTH  WORDS  ALLOT  CREATE  DEFER  VARIABLE  AGAIN  AND
REPEAT BEGIN  BL      #      C,      DO      THEN  UNTIL  ELSE  U>  U<=
U>=   U<      WHILE  XOR    X,      XC,     HERE  IF    <=   <    LOOP
,,    MD      >      >=   NOT    OUT    OR    ?<RESOLVE  ?<MARK ?>RESOLVE
?>MARK 0>=   0<    0<>   0=    SED    DUMP  (
```

```
Im System sind : 50 doppelt definierte Worte / Namen ok
---[ ZF-Forth LOG-File ]----(geschlossen)-----21.05.1994--10:48:48---
```

gleich ist. Das macht bei 1241 Worten rund 20 % aus. Dieser „Hashalgorithmus“ (MHASH) ist offensichtlich sehr unefektiv...

Nach Hannes' Forderung, konkreten Code sehen zu wollen, und nach den bisherigen Mühen (die eigentlich gar nicht beabsichtigt waren), bin ich selbst auf „konkreten Code“ neugierig geworden.

Ich habe mir überlegt, daß ich, neben diversen Mischformen, eigentlich nur zwei Alternativen habe, den in THREAD2. SEQ vorgestellten Hashalgorithmus zu verbessern. Die erste Möglichkeit wäre, den Algorithmus selbst so weit zu verbessern, daß grundsätzlich keine Mehrfachnennungen mehr auftreten können. Das hätte langwieriges Experimentieren bedeutet, unzählige Test, zu denen noch nicht einmal Routinen existieren, und einen „Zeitverbrauch“, den ich im Augenblick nicht aufbringen mag.

Die zweite Alternative ist eine „Kollisionsstrategie“, die mit den Mehrfachnennungen möglichst einfach zurecht kommt. Lineare Listen, wie sie das ZF selbst führt, wollte ich nicht implementieren. Es soll doch alles etwas anders werden...

Wenn man sich aber ansieht, daß die Mehrfachnennungen sich, bis auf ganz wenige Ausnahmen, auf doppelte Nennungen konzentrieren, dann drängt sich fast automatisch der Gedanke auf, daß „soviel nicht passieren kann“. Um es kurz zu machen - ich verzichte sogar auf eine ausgefeilte Kollisionsstrategie, und gehe davon aus, daß in der Hashtabelle genügend Platz vorhanden sein soll, um auch diese „Fehlschläge“ aufzunehmen. Mit anderen Worten, ich lege fest, daß jeder Funktionswert, der in der Tabelle einen besetzten Platz vorfindet, einfach in den nächsten freien Platz abgelegt werden soll. Ich lasse also sowohl Primär-, als auch Sekundärkollisionen einfach zu !

```
: .DoppelteWorte ( -- )
0 Zaehler !
CR CR
HERE 500 + \ Start des Puffers
BEGIN
DUP COUNT + DUP >R >R DUP \ Adresse vom
\ nächstem Wort auf TOR
BEGIN
DUP C@ R@ C@ = \ vergleiche die COUNTs
IF DUP COUNT R@ COUNT \ wenn gleich,
\ vergleiche Strings
SEARCH NIP
IF DUP COUNT TYPE TAB \ wenn gleich,
\ Wort ausgeben
Zaehler INCR \ und Zähler
\ inkrementieren
THEN
THEN
R> COUNT + DUP >R C@ \ nächstes Wort
\ 'aufnehmen'
0= UNTIL \ Bei COUNT = 0 ist ENDE
R> DROP \ Stack und Returnstack 'säubern'
DROP DROP
R> DUP C@ \ solange, bis ein COUNT '0' ist
0= UNTIL DROP
CR CR ." Im System sind : " Zaehler @ 5 .R
." doppelt definierte Worte / Namen" ;

\ =====

: Hashtest ( -- )
DARK
CR
CR ." ...statistische Infos zum Dictionary..."
CR
HERE 500 + =: Puffer \ mittlere Wortlänge
\ errechnen
Puffer 10000 ERASE
0 =: Flag
Voclink.Abarbeiten
?WortLaenge
Tabelle 5000 ERASE \ Mehrfachnennungen
\ auszählen
Nenner 20 ERASE
1 =: Flag
```



Worte am (seidenen) Faden

```
Voclink.Abarbeiten
?Mehrfachnennungen
.DoppelteWorte ; \ welche Namen sind
                  \ mehrfach definiert ?
```

Die Worte des ZF-Directories, bzw. die WortHeader, sind alle im YSEGment abgelegt. Wie ich in den vorhergegangenen Files zeigen konnte, beinhaltet diese Ablage den COUNT des Namens, den Namenstext und die CFA des entsprechenden Wortes. Es könnte durchaus reichen, wenn in der Hashtabelle an den Plätzen der Funktionswerte die Segmentadressen der Header, sprich: die NFAs der einzelnen Worte festgehalten würden. Dabei würde aber nicht mehr berücksichtigt werden, welchem Vokabular ein Wort zugeordnet ist. Ich habe mich deshalb entschlossen, ein 'komplett neues HASHING' aufzubauen.

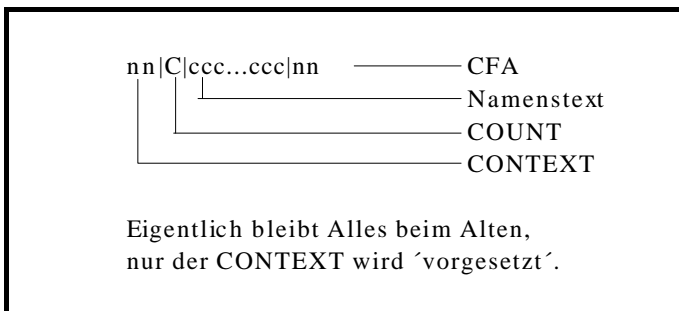
Konkretes Hashing

Ich möchte „ein komplett neues Hashing“ programmieren. Meine Version soll zumindest nicht „schlechter“ werden als jene, die bereits im ZF und im F-PC (warum sollte Tom Zimmer das im F-PC anders gemacht haben?) definiert ist. Ich will also auch weiterhin eine „kontextbezogene“ Suche im Dictionary ermöglichen.

Die Hashtabelle selbst will ich im aktuellen Segment halten, also im CODE-Segment des ZF. Die Header sollten aber, schon aus Platzgründen, in einem separaten, von DOS anzufordernden Segment untergebracht sein. Nach allen bisherigen Vorarbeiten muß ich mir nur noch überlegen, wie die Header aufgebaut sein sollen. Danach kann's schon losgehen. Der folgende Code zeigt also „konkretes Hashing“, mit „allem was dazugehört“ - selbstverständlich in HighLevel FORTH.

So sind „meine“ Header strukturiert.

```
\ THREAD3.SEQ - EXPERIMENTE zu ZF's HASHING
\
\ Friederich Prinz, Forthgruppe Moers, Mai '94
\ -----
```



```
CREATE      HashTabelle 10000 ALLOT
\ Platz für 5000 Hash-Einträge

0 CONSTANT  HSegment
\ Segment für 'gehashte' Worte

1 CONSTANT  HHere
\ Offset auf das HashSegment, Zeiger der Endemarke

0 CONSTANT  VocLink      \ CONTEXT
```

```
0 CONSTANT  MCount
\ HilfsSpeicher, siehe Definition...

0 CONSTANT  EndeFlag

0 CONSTANT  StringAdresse

\ -----

: Segmente.Anfordern ( -- )
\ Speicher von DOS anfordern
2048 ALLOC =: HSegment ;
\ 32768 Byte für das HashSegment

: Segmente.Freigeben ( -- )
HSegment DEALLOC DROP ;

: !Definition ( NFA -- ) \ Definition 'ablegen'
VocLink HSegment HHere !L \ 'Kontext' ablegen
2 +!> HHere \ HHere + 2
  DUP YC@ 31 AND 3 + \ COUNT + 3
  YSEG @ ROT HSegment HHere 4 PICK
\ ab NFA Eintrag kompl. ins HashSeg.
  CMOVE!
  +!> HHere ;
\ HHere entsprechend 'hochsetzen'

: Haschen ( NFA HF.Wert -- )
\ Eintrag in die Hashtabelle erzeugen
HashTabelle +
\ Adresse in der Tabelle laut HASHING
BEGIN
  DUP @ \ wenn die Adresse belegt ist, dann
  0= IF FALSE \ suche den nächsten freien Platz...
  ELSE 2+ TRUE
  THEN
  0= UNTIL \ der Offset auf das HashSegment,
  HHere SWAP ! \ HHere, wird in der Tabelle vermerkt
!Definition ; \ Die Definition ins HSegment kopieren

: Mhash ( NFA -- HashFunktionswert NFA )
\ alles Andere ist nur das 'Drumherum' !!!
\ =====
DUP >R
  DUP 1+ YC@ 31 AND SWAP
  DUP YC@ 31 AND 1+ 1 DO 1+ DUP YC@ 31 AND
  I * ROT + SWAP
  LOOP
  DROP 16 *
  10000 MOD
  R> SWAP ;

: Thread.Abarbeiten ( Thread -- )
\ in THREAD1.SEQ beschrieben
BEGIN
  DUP L>NAME
  Mhash
  Haschen
  Y@ DUP
  0= UNTIL
  DROP ;

: Alphabet.Abarbeiten ( Thread -- )
\ in THREAD1.SEQ beschrieben
ASCII A #THREADS 0
DO DUP PAD 1+ C! PAD 2 PICK HASH @
  DUP 0<> IF Thread.Abarbeiten
  ELSE DROP
  THEN
  1+
LOOP DROP DROP ;

: VocLink.Abarbeiten ( -- )
\ in THREAD1.SEQ beschrieben
1 PAD C! VOC-LINK @
BEGIN
\ -----
```



```

    DUP =: VocLink
\ der CONTEXT soll in die Header aufgenommen
\ werden...
\ -----
    DUP #THREADS 2 * -
    Alphabet.Abarbeiten
    @ DUP
0= UNTIL DROP ;

\ =====
: HashStart ( -- )
    HashTabelle 10000 ERASE
    Segmente.Anfordern
    HSegment 0 32768 0 LFLILL
    Voclink.Abarbeiten ;

\ -----
    HASHSTART

\ Das kann 'vorab' schon mal ausgeführt werden...
\ -----
: MChash ( Str_Adr -- HF.Wert NFA )
    \ ...wie MHash, erwartet die Adresse
    \ jedoch im aktuellen CodeSegment,
    DUP >R \ also KEINE NFA !
    DUP 1+ C@ 31 AND SWAP
    DUP C@ 31 AND 1+ 1 DO 1+ DUP C@ 31 AND
        I * ROT + SWAP
        LOOP
    DROP 16 *
    10000 MOD
    R> SWAP ;

: ?Context ( HS_Adr -- f )
    \ steht das Wort im Context des aktuellen
    \ Vokabulars ?
    HSegment SWAP @L CONTEXT @ 32 + = ;

: ?Count ( HS_Adr -- f )
    \ ist der COUNT gleich dem COUNT des zu prüfenden
    \ Wortes
    2+ HSegment SWAP C@L 31 AND MCount = ;

: ?String ( HS_Adr -- f )
    2+ HSegment SWAP ?CS: PAD MCount CMOVELE
    \ sind die Zeichenketten gleich
    StringAdresse 1+ PAD 1+ MCount 1- COMPARE ;

: @CFA ( HS_Adr -- CFA )
    \ hole die CFA des gefundenen Wortes
    2+ DUP HSegment SWAP C@L 31 AND 1+ +
    HSegment SWAP @L ;

: FindeWort ( -- CFA | f ) \ übernehme einen Namen
    BL WORD \ aus dem Eingabestrom, und sieh
    COUNT =: MCount 1- \ nach, ob dieser Name
    Mchash \ 'gehasht' wurde...
    SWAP =: StringAdresse
    FALSE =: EndeFlag
    HashTabelle +
    BEGIN
        DUP @
        DUP 0= IF DROP \ wenn zum HashFunktionswert
        ." ...nicht definiert !" \ kein Eintrag in der
        \ HashTabelle vorhanden ist, dann ist das Wort
        EXIT \ auch nicht definiert
        THEN \ definiert...
    DUP ?Context \ CONTEXT
    IF DUP ?Count \ COUNT
        IF DUP ?String \ STRING vergleichen
            0= IF @CFA
                TRUE =: EndeFlag
            ELSE DROP 2+
            THEN

```

```

    ELSE DROP 2+
    THEN
    ELSE DROP 2+
    THEN
        EndeFlag
    TRUE = UNTIL
    NIP ;

```

... ich hab's natürlich ausprobiert: VOR dem Kompilieren dieser Datei habe ich im Vokabular FORTH definiert,

```

: FRITZ42 EMIT ;          und im Vokabular BUGS,
: FRITZ 7 EMIT ;

```

Anschließend habe ich das File übersetzen lassen und zunächst in FORTH aufrufen:

```
FindeWort Fritz EXECUTE * ok
```

...und in BUGS

```
FindeWort Fritz EXECUTE (PIEP) ok
```

... es funktioniert also. Wer hätte das gedacht ? Ich selbst am allerwenigsten ;-)

„Nachwort“

Mit einer Frage nach dem HASHING hat etwas angefangen, was mich tief in ZF's „Innereien“ geführt hat. Ich fand es sehr aufregend und bin dem Hannes dankbar für den Anstoß. Jetzt kann ich nur hoffen, daß ihm das alles „konkret“ genug ist. Mir selbst reicht es allemal. Aber schließlich hat Johannes Teich nicht nur die Aufgabe gestellt, sondern mich in unseren Diskussionen via DFÜ immer wieder „auf den richtigen Weg“ gebracht. Auch Dirk Zoller, dem die Forthgemeinde das PFE verdankt, hat mit Denkanstößen seinen Teil zum Gelingen meiner Experimente beigetragen, ebenso wie Zbigniew Dianyszin, dessen Verdienst es ist, daß ich diesen „Bericht“ geschrieben habe. Zbigniew ist davon überzeugt, daß sowohl die Ausflüge in ZF's „Innereien“, als auch die generelle Aufarbeitung des Themas HASHING die Leser der VD brennend interessiert. HASHING ist etwas, dessen Namen die Meisten gehört haben, die Wenigsten richtig schreiben können und kaum Jemand jemals „in konkretem Code“ gesehen, geschweige denn selbst entwickelt hat. Nun, das „Entwickeln“ können sich die Leser des Berichtes zukünftig sicher sparen - es sei denn, Jemand muß Ähnliches in irgendeinem System implementieren...

Gleichzeitig hoffe ich natürlich auch, daß allen Lesern der Bericht ausführlich genug ausgefallen ist, ohne langweilig zu werden (trotz der vielen redundanten Definitionsteile). Ich habe mich diesmal bewußt nicht an konventionelle Aufsatzstrukturen gehalten, sondern versucht, die Texte soweit wie möglich in der Form wiederzugeben, wie sie durch das Netzwerk (Z-NETZ) gegangen sind. Denn wenn Sie „das Hashing“ trotz aller meiner Bemühungen noch nicht so ganz verstanden haben sollten, dann finden sich (unter anderem) in *de/comp/lang/forth* bestimmt jede Menge FORTHer, die Ihnen gerne und bereitwillig mehr dazu erzählen.



Ein letzter Gedanke

Und wozu das Alles ? Ein „geschicktes“ Hashing beschleunigt die Suchläufe im Wörterbuch. Das macht das Kompilieren schneller. Wieviel schneller ? Ich hab's nicht ausprobiert. Auf das Kompilat schließlich, sprich: auf die Laufzeit, hat das Hashing keinen Einfluß. WORDS jagt den Inhalt der Vokabulare auf meinem Rechner ohnehin viel zu schnell über den Bildschirm. Da muß nichts schneller werden. Ich habe halt ein bißchen was über das ZF gelernt, über Strukturen in FORTH-Systemen und sogar über ein „Sortierverfahren“ mit dem Namen Hashing...

fep

Fortsetzung von Seite 8

'vertreten', dann sind's drei.

Was bringt es dem Verein, wenn ihr wichtigstes Produkt -- ich übertreibe mal etwas a la Rafael -- auch ohne Beitrag komplett erhältlich ist und z.B. auf CD zudem noch einfachst transportiert werden kann.

Arndt Klingelberg

Betreff: Re: Vierte Dimension im Netz

Na natürlich die Gemeinnützigkeit :-)

(siehe Satzung <http://www.forth-ev.de/statute.htm>)

Egmont Woitzel

Betreff: Re: Vierte Dimension im Netz

Hi, Arndt (akg@aachen.forth-ev.de)

> ...VD in INTER...

> Was bringt es dem Verein, wenn ihr wichtigstes Produkt

...

Das Internet ist passiv. Zusenden einer Vereinspostille ist aktiv seitens des Vereins - er erreicht und kontaktet periodisch seine Mitglieder und bringt sich in Erinnerung.

PS: Gerne würde ich in der 4D Hinweise auf interessante URL's lesen wollen. Könnte die Redaktion das nicht mal starten und alle dazu aufrufen? (Fritz, du liest doch mit, oder?)

Michael Kalus

Ich lese mit, gelegentlich – weil mir zu mehr die Zeit fehlt – und ‚fische‘ dann solche Ketten aus dem Netz. Die Idee gefällt ! Also schickt der VD alle interessanten URLs, die ihr kennt.

Und weil wir gleich bei der Arbeit sind – wer findet sich bereit, das Geschehen in de.com/lang/forth zu beobachten und der VD 4 Mal pro Jahr eine Zusammenfassung zu senden ?

fep

Ein Programm zum Knacken von polyalphabetischen Codes (Teil 1)

Hugh Aguilar <haguilar@dancris.com>, Phoenix, USA

Mit freundlicher Genehmigung der amerikanischen Forth Interest Group übersetzt von Fred Behringer, München. Der Originalaufsatz erschien in der Forth Dimensions, Band XX, Nummer 5, 6, vom Januar/April 1999.

Das hier beschriebene Chiffrierverfahren ist uralte. Es stammt aus dem 15. Jahrhundert und wurde im Amerikanischen Bürgerkrieg im 19. Jahrhundert ausgiebig verwendet. Der Autor baut sich einen "Computer aus Pappkarton", sozusagen, in Forth natürlich, und zeigt, wie man solche Codes knacken kann. Auf die Buchstabenhäufigkeiten kommt es an. (Wenn die Stabsoffiziere damals Forth gehabt hätten, wäre der Krieg vielleicht anders ausgegangen (?))

Stichworte: Kryptographie, automatische Dechiffrierung, Buchstaben-Häufigkeitsanalyse

Zur Übersetzung: Die Sprache des Originals, Englisch, wird in diesem Artikel in dreierlei Hinsicht verwendet: (1) im Befehlssatz der Computersprache, hier in Form von Forth-Worten, (2) im Text, der analysiert, chiffriert und dechiffriert werden soll, (3) als Verständigungsmittel zwischen Autor und Leser. Forthworte zu "übersetzen" (1), wäre unvernünftig, ungebrauchlich und unprofessionell. Die zu analysierenden Texte neu zu fassen und "durch die Maschine zu jagen" (2), wäre aufwendig, fehlerträchtig und kaum förderlich. Ich beschränke mich als Übersetzer auf (3).

Über den ganzen Text des Artikels ziehen sich die Worte **PLAINTEXT** (Klartext) und **CIPHERTEXT** (Geheimtext) hin. Sie bezeichnen Datenpuffer, die den unverschlüsselten und den irgendwie schon verschlüsselten Text (oder deren Zwischenprodukte) enthalten. Zuweilen wird auch in den Kommentaren von **PLAINTEXT** und **CIPHERTEXT** gesprochen. Ich behalte das bei, um den Zusammenhang nicht zu stören.

PolySub - bekannt, aber nicht sehr sicher. In diesem Artikel erklären wir das Chiffrierverfahren nach der polyalphabetischen Substitution, das den meisten Lesern bekannt sein wird. Das ist jenes Verfahren, bei dem ein bestimmtes Schlüsselwort wiederholt mit dem Klartext geXORT wird, um den Geheimtext zu erzeugen (oder umgekehrt). Wir stellen dann ein Programm vor, das diesen Code knackt, unter der Voraussetzung allerdings, daß die Buchstaben und Zeichen im Klartext mit unterschiedlicher Häufigkeit auftreten und daß insbe-



sondere ein bestimmtes Zeichen wesentlich häufiger erscheint als alle anderen (wir nehmen an, daß das der Zwischenraum ist). Der vorliegende Artikel wendet sich an Anfänger. Wir erklären daher unser Codeknackprogramm auf der Implementations-ebene anhand von vielen Programmbeispielen.

Auf dem Computer wird PolySub normalerweise über **XOR** implementiert. Damit kann ein und dasselbe Programm sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet werden (man beachte, daß **XOR** sich selbst rückgängig macht: $a \oplus b \oplus b \oplus a = a$). In der Zeit vor Erfindung des Computers hat man Plus und Minus verwendet. PolySub wurde von Leon Battista Alberti (1404-1472) erfunden und im Amerikanischen Bürgerkrieg von der Unionsarmee ausgiebig verwendet.

Werfen wir einen Blick auf ein Beispiel für PolySub, bei welchem Plus und Minus verwendet werden. Als Alphabet setzen wir alle Großbuchstaben in der Codierung 0 bis 26 ein, mit 0 als Zwischenraum. Als Schlüssel soll "DOOR" genommen werden (siehe Abbildung 1).

Zur Chiffrierung verwenden wir Modulo-Addition. Dechiffriert werden kann die Nachricht dann über Modulo-Subtraktion. Das ist eine ausgezeichnete Vorgehensweise für jeden, der keinen Zugang zu einer elektronischen Rechenmaschine hat. Man kann sich nämlich für den vorliegenden Zweck leicht einen "Computer" aus Pappkarton bauen.

Zeichnen Sie sich dazu auf Karton einen Kreis und schneiden Sie diesen aus. Machen Sie in der Mitte ein Loch und heften Sie die Scheibe (über eine Niete) an eine ähnliche zweite Scheibe, so daß Sie die erste über der zweiten drehen können. Auf beiden Scheiben sind die Buchstaben im Uhrzeigersinn aufgeschrieben. Zur Chiffrierung oder Dechiffrierung nimmt man den momentanen Buchstaben des "Verschlüsselungsstroms" (des Stroms von aneinandergereihten Wiederholungen des Schlüsselwortes) auf der inneren Kartonscheibe und plaziert ihn über den Zwischenraum auf der äußeren Scheibe. Beim Chiffrieren sucht man sich den gerade betrachteten Buchstaben der Nachricht auf der äußeren Scheibe heraus und findet den dazugehörigen verschlüsselten Buchstaben am entsprechenden Platz auf der inneren Scheibe. Beim Dechiffrieren sucht man sich den gerade betrachteten Buchstaben der chiffrierten Nachricht auf der inneren Scheibe heraus und findet den dazugehörigen Buchstaben der unverschlüsselten Nachricht auf der äußeren Scheibe. Man beachte, daß die berühmte Verschlüsselung nach Julius Caesar (zu jedem Buchstaben drei hinzuzählen) einfach nur eine abgespeckte Form des Plus-Minus-Schemas ist. Ihr Schlüssel ist lediglich einen Buchstaben lang ("C"). Die Verschlüsselung nach Julius Caesar ist eine "Monoalphabetische Substitutionsverschlüsselung".

Die Sicherheit des PolySubs kann durch Verlängerung des Schlüssels noch etwas erhöht werden. Das läßt sich am einfachsten durch wiederholte Verschlüsselung der Nachricht er-

reichen. Wenn Ihr Schlüssel z.B. "DOORFENCE" lautet, beträgt die Schlüssellänge neun. Wenn dagegen die Nachricht erst mit "DOOR" und dann mit "FENCE" verschlüsselt wird, ist die Wirkung dieselbe, als wenn nur ein einziges Mal, aber mit einem Zwanzig-Zeichen-Schlüssel (4*5) verschlüsselt wird, nämlich mit "JTCUIUTFGTUWRRTXICRW". Die Sicherheit ist sogar noch etwas höher, da der insgesamt wirkende Zwanzig-Zeichen-Schlüssel einen Buchstabensalat darstellt und nicht so leicht zu erraten ist wie "DOORFENCE", das sich erkennbar aus Wörtern der englischen Sprache zusammensetzt. Wenn man solche Mehrfach-Verschlüsselungen vornimmt, sollte man darauf achten, daß keine der Schlüssellängen gemeinsame Nenner haben. Haben die Schlüssel dieselbe Länge, wie z.B. "DOOR" und "GATE", dann ist es in seiner Wirkung immer noch ein Vier-Zeichen-Schlüssel (wenn auch die Buchstaben zumindest durcheinandergewürfelt werden, hier nämlich zu "KPIW").

Der erste Teil unseres CrakPoly-Programms enthält den Code zum Laden und Abspeichern der Dateien und zu ihrer Chiffrierung und Dechiffrierung. Danach gehen wir zur Entzifferung von chiffrierten Texten über, für die wir keinen Schlüssel besitzen. Beim Knacken des PolySub-Codes unterscheiden wir zwei Phasen: zunächst die Ermittlung der Schlüssellänge und dann die Bestimmung des Schlüsselinhaltes.

Vorbereitende Programmteile - Dateien chiffrieren und dechiffrieren

Unser PolySub-Codebrechprogramm heißt "CrakPoly.scr" und wurde in UR/Forth, das von der Firma Laboratory Microsystems Inc. stammt, geschrieben. Der Quelltext befindet sich in Abbildung 2. CrakPoly sollte auf allen Forth-83-Compilern laufen. Es wurde sowohl unter einem 32-Bit- als auch unter einem 16-Bit-UR/Forth getestet. Dem Leser wird anempfohlen, **QI** (das auf Screen 5 bereitgestellt wird) an verschiedenen Stellen in den Worten einzubauen, um das Programm etwas aufzuspalten. Die Ausführung wird unterbrochen und der Anwender kann sich den Inhalt seiner Variablen ansehen, bevor er mit dem Programm weitermacht.

Wir haben zwei Datenpuffer vorgesehen, **CIPHERTEXT** und **PLAINTEXT** (Geheimtext und Klartext). Jedem von beiden sind **FILE_SIZE** Bytes an Speicherplatz zugeordnet. **FILE_SIZE** wird in Screen 1 definiert und wurde zunächst auf einen verhältnismäßig kleinen Wert gesetzt, damit auch Leser mit 8-Bit-Computern die Programme einlesen und zum Laufen bringen können. Leser mit 32-Bit-Computern sollten **FILE_SIZE** auf einen größeren Wert setzen.

Das Wort **INPUT_FILE** in Screen 17 dient zum Einladen einer Datei in den Speicher. Es nimmt zwei Parameter auf, den Dateinamen und den Puffer-Zeiger. Der Dateiname sollte die Adresse einer Zeichenkette mit Längenangabe (counted string) sein, die den voll ausgeprägten Dateinamen enthält. Der Puffer-Zeiger sollte entweder **CIPHERTEXT** oder **PLAINTEXT** sein. **OUTPUT_FILE** befindet sich ebenfalls in Screen 17 und übernimmt ebenfalls den Dateinamen und



den Puffer-Zeiger, gibt aber den Inhalt des Puffers auf die Datei aus.

Wenn sich in **PLAINTEXT** ein Dokument befindet, dann bewirkt die Ausführung des Wortes **ENCRYPT** in Screen 14, daß sich **CIPHERTEXT** mit der verschlüsselten Version des Dokumentes füllt. Führt man das Wort **DECRYPT** aus, das sich ebenfalls in Screen 14 befindet, dann wird das Dokument in **CIPHERTEXT** entschlüsselt und **PLAINTEXT** füllt sich mit der entschlüsselten Version.

Man beachte, daß **ENCRYPT** und **DECRYPT** von den Worten **LOW_ENCRYPT** und **LOW_DECRYPT** Gebrauch machen, welche in Screen 4 definiert werden. Diese Worte in Screen 4 sind für die **XOR**-Version von PolySub bestimmt. In den Screens 2 und 3 stellen wir noch andere Versionen von **LOW_ENCRYPT** und **LOW_DECRYPT** bereit. Beide Screens sind als Kommentare gefaßt (auskommentiert). Screen 2 bezieht sich auf das Plus-Minus-PolySub, Screen 3 auf das Minus-Plus-PolySub. Wenn jemand eines davon verwenden möchte, möge er Screen 4 auskommentieren und stattdessen 2 oder 3 compilieren.

Phase 1 --- Bestimmung der Schlüssellänge

Zur Ermittlung der Länge des Schlüssels gehen wir davon aus, daß die Zeichen im Klartext mit unterschiedlicher Häufigkeit auftreten. Uns interessiert dabei nicht, welche Zeichen häufiger auftreten als andere und wie ihre Verteilung aussieht, vorausgesetzt, sie unterliegen keiner Rechteckverteilung. Wir werden den Geheimtext wiederholt weiterschieben und ihn mit dem nicht verschobenen Geheimtext vergleichen. Wir zählen dann, wieviele der miteinander verglichenen Zeichen mit den Zeichen im nicht verschobenen Teil übereinstimmen.

Wir haben ein Datenfeld (array) namens **COINCIDENCES** vorgesehen. Der erste Index liefert die Anzahl der Übereinstimmungen bei Verschieben des Geheimtextes um ein Zeichen, der zweite bei Verschieben um zwei Zeichen usw. **COUNT_COINCIDENCES** in Screen 18 zählt die Übereinstimmungen. **COINCIDENCES** enthält Prozentanteile, nicht die eigentlichen Anzahlen, da ja bei jedem Aufruf von **COUNT_COINCIDENCES** immer wieder eine andere Zahl von Vergleichen angestellt wird. Die Prozentangaben enthalten zwei Nachkommastellen.

FILL_COINCIDENCES in Screen 19 ruft **COUNT_COINCIDENCES** wiederholt auf und füllt das Datenfeld **COINCIDENCES**. Im Wort **SEARCH_SIZE** wird festgehalten, wieviele Verschiebungen wir vornehmen. Ist die Datei klein, führen wir nur ein Drittel der Gesamtzahl durch. Je mehr wir verschieben, desto geringer wird nämlich die Genauigkeit. Würden wir uns über die gesamte Dateilänge hinweg bewegen, wären die Zahlen am Ende nur noch Müll und würden alles zunichte machen. Die Art, wie **COUNT_COINCIDENCES** vom Autor ursprünglich konstruiert war, stellte sich als unbrauchbar heraus. Der Geheim-

text sollte nämlich so im Kreise herumgedreht werden, daß die Zeichen am Ende der Datei mit denen am Anfang zum Vergleich kämen. Die Genauigkeit würde dann mit steigender Zahl von Verschiebungen zwar nicht abfallen, aber die Idee war trotzdem schlecht, da dann übereinstimmende Zeichen mehrfach gezählt und die Zahlen einander angeglichen wurden.

Screen 21 enthält das Wort **SHOW_COINCIDENCES**, das die eben besprochenen Worte dazu benutzt aufzuzeigen, was im Datenfeld **COINCIDENCES** ist. Verwendet man **SHOW_COINCIDENCES**, um sich **COINCIDENCES** anzusehen, so wird man in den Werten Spitzen entdecken. Diese Spitzen treten an den Stellen von Vielfachen der Länge des zur Chiffrierung verwendeten Schlüssels auf. Richtet man also sein Augenmerk auf **COINCIDENCES**, so kann man die Schlüssellänge leicht herausbekommen.

Wir wollen aber, daß unser Programm das automatisch erledigt. Einfach ist das nicht, da man nicht genau sagen kann, welche Schwelle ein Wert überschritten haben muß, um als Spitze gewertet zu werden. Dieser Schwellwert ist von den Daten abhängig. Zudem ist es ziemlich egal, wie sorgfältig der Schwellwert ausgewählt wurde, es werden immer Werte über der Schwelle liegen, die gar keine Spitzen sind, und solche unter ihr, die eigentlich als Spitzen hätten gewertet werden sollen. Die Daten sind sehr unterschiedlich, besonders wenn kleinere Dateien geknackt werden sollen.

Wir setzen unseren Schwellwert auf das Mittel zwischen Maximum und Minimum der Daten in **COINCIDENCES**. Das wird von **CALC_TRESHOLD** in Screen 22 erledigt. Ursprünglich hatten wir es mit einem konstanten Wert von 4% versucht. Das funktionierte nicht, da der Schwellwert je nach Schlüssellänge verschiedenhoch ausfällt. Dann versuchten wir es mit dem Durchschnittswert. Das ging ebensowenig. Der war viel zu klein, besonders bei größeren Schlüssellängen, und lieferte uns eine Menge falscher Spitzen. Als nächstes probierten wir dann den Durchschnittswert plus der Standardabweichung, multipliziert mit einer empirisch ermittelten Konstanten, aus. Wird die Konstante beispielsweise zu 0,68 gewählt, so bleiben 75% der Werte unter dem Schwellwert. Das klappte besser, war aber viel zu kompliziert und immer noch nicht gut genug.

Am besten und einfachsten ging es mit dem Mittel aus Maximum und Minimum. Wir bekommen Spitzen, die sich um einen bestimmten höheren Wert herum gruppieren, und Nichtspitzen um einen bestimmten niedrigen Wert herum. Es bilden sich mehr Nichtspitzen als Spitzen heraus, besonders wenn die Schlüssellänge groß ist. Und das war es auch, was uns unsere Ergebnisse bei dem Versuch mit dem Durchschnitt verdarb. Diese Ungleichheit hatten wir mit der Standardabweichung zu kompensieren versucht. Wenn wir das Mittel aus Maximum und Minimum nehmen, brauchen wir uns über das Verhältnis von Spitzen zu Nichtspitzen keine Gedanken zu machen. Das Mittel zieht eine Gerade zwischen



höchstem und niedrigstem Wert und durch diese Gerade werden die Spitzen ziemlich gut von den Nichtspitzen getrennt. **CALC_TRESHOLD** braucht nicht perfekt zu sein, da das Datenfeld **KEY_LENGTHS**, das als nächstes beschrieben werden soll, Fehler ausbügelt, die durch irrtümlich als Nichtspitzen genommene Spitzen (und umgekehrt) ausgelöst werden (solange die Zahl der Fehler nicht zu groß wird).

Wir haben da also noch ein Datenfeld namens **KEY_LENGTHS**, so groß wie die maximal vorgesehene Schlüssellänge, das wir mit den in Prozent angegebenen Wahrscheinlichkeiten füllen, mit denen ein Schlüssel bestimmter Länge auftritt. Das müssen wir tun, weil es wegen der schon erwähnten Schwankungen keinen Weg gibt, die Schlüssellänge mit absoluter Sicherheit vorauszusagen. Dieses Datenfeld wird über **FILL_KEY_LENGTHS** in Screen 23 gefüllt, das die Abstände zwischen den Spitzen zählt. Wären die Abstände alle gleich, dann wären wir absolut sicher, daß das die Schlüssellänge ist. Gewöhnlich sind sie das aber nicht. Also zählen wir ab, wie oft die einzelnen Abstände auftreten.

Die Ergebnisse dieser Zählungen werden nach **KEY_LENGTHS** gelegt. **KEY_LENGTHS%** in Screen 24 wandelt die Zählergebnisse in Prozentangaben um, und zwar hauptsächlich aus ästhetischen Gründen für die spätere Darstellung am Bildschirm. Für **CALC_KEY_LENGTHS** hätten wir das nicht gebraucht. Wir haben außerdem noch eine Variable namens **BIG_KEY_LENGTHS** vorgesehen, die solche Spitzenabstände zählt, die wegen ihrer Größe nicht in **KEY_LENGTHS** passen. Sie bleibt hoffentlich Null.

CALC_KEY_LENGTH berechnet die eigentliche Schlüssellänge. **KEY_LENGTHS** wird von diesem Wort zunächst aufgefüllt und dann nach dem größten Wert durchsucht. Der zu diesem Wert gehörige Index ist unsere Schlüssellänge. Wenn wir zwei oder mehr Werte haben die gleich sind, nehmen wir den kleinsten Index. In so ziemlich allen Fällen, da das auftritt, ist der höhere Index ein Vielfaches des niedrigeren. Der kleinste Index ist die eigentliche Schlüssellänge (sonst hätten wir einen Schlüssel, der aus einer Zeichenkette bestünde, die eine gewisse Anzahl von Malen wiederholt wird).

Screen 25 enthält **FILL_KEY_LENGTH**. Von diesem Wort wird alles erledigt, was zur Bestimmung der Schlüssellänge nötig ist. Das ist das Wort, das der Anwender über die Tastatur eingibt, um Phase 1 des Programms zu bearbeiten. Man beachte, daß der Anwender die Schlüssellänge über **KEY_LENGTH!** manuell einstellen kann, wenn er mit dem, was das Programm für die Schlüssellänge hält, nicht einverstanden ist. **FILL_KEY_LENGTH** zeigt den höheren Anteil von **COINCIDENCES** oben am Bildschirm an. Diese Rohdaten sind nur am Rande nützlich. **FILL_KEY_LENGTH** zeigt **KEY_LENGTHS** unten am Bildschirm an. Hier kann der Anwender sehen, mit welchen Wahrscheinlichkeiten die einzelnen Schlüssellängen auftreten. Das gibt ihm einen An-

halt dafür, mit was er in **KEY_LENGTH!** hineingehen soll, wenn er nicht damit einverstanden ist, was das Programm für die wahrscheinlichste Schlüssellänge herausgefunden hat. In der Praxis wird das kaum nötig sein. **FILL_KEY_LENGTH** findet fast immer den richtigen Wert.

Wir haben jetzt das Programm zur Bestimmung der Schlüssellänge besprochen. Die Schlüssellänge brauchen wir, bevor wir den Schlüsselinhalt bestimmen können. Das Problem, den Schlüsselinhalt zu bestimmen, werden wir in Teil 2 unseres Artikels in Angriff nehmen. Dieses Problem wird unsere Programmierkünste weniger in Anspruch nehmen als das Programm zur Bestimmung der Schlüssellänge. Die Bestimmung des Schlüsselinhalt wird aber im Unterschied zum hier besprochenen ersten Programmteil interaktive Hilfe von seiten des Benutzers erfordern.

Abbildung 1

MEATLOAF FOR DINNER
 - der Klartext (die unverschlüsselte Nachricht)
DOORDOORDOORDOORDOO
 - der Verschlüsselungsstrom

QTPKPCPX DUCIDSXERTF
 - der Geheimtext (die verschlüsselte Nachricht)

Untenstehend dasselbe in Zahlencodierung

Abbildung 2

13 05 01 20 12 15 01 06 00 06 15 18 00 04 09 14 14 05 18
04 15 15 18 04 15 15 18 04 15 15 18 04 15 15 18 04 15 15
17 20 16 11 16 03 16 24 04 21 03 09 04 19 24 05 18 20 06

Screen 0
 \ CRAKPOLY 19.36 29.05.99

Brechen der polyalphabetischen Substitutionschiffre (XOR)
 Verfaßt von Hugh Aguilar
 Januar/Februar/März/April 1999 Forth Dimensions

Bemerkung des Übersetzers: Ich habe im Programm (wenn, dann sowieso nur in den Bildschirm-Anzeigestrings) so gut wie keine Veränderungen vorgenommen. Man störe sich also nicht am englischen Erscheinungsbild. Die Kommentare nach den nach links gerichteten Schrägstrichen habe ich möglichst alle ins Deutsche übertragen.

Aus dem Archiv: Skytale von Sparta, ca. 500 v. Chr., Transposition von Zeichen, beschriebener Papierstreifen auf einem Zylinder.



Screen 1
 \ 32-Bit-Arithmetik CHARS MOSTEST 20.33 30.05.99

```
WSIZE CONSTANT W \ bequemer einzutippen

\ Die Worte (W+, W-, W* und W/) sind für ein
\ 32-Bit-System vorgesehen. Wird ein 16-Bit-System
\ verwendet, müssen sie umgeschrieben werden.
: W+ 4 + ;
: W- 4 - ;
: W* 2* 2* ;
: W/ 2/ 2/ ;
```

```
256 CONSTANT CHARS

CREATE MOSTEST 0 , BL MOSTEST C!
\ häufigstes Klartextzeichen

5000 CONSTANT FILE_SIZE \ maximale Dateilänge
```

Screen 2
 \ LOW_ENCRYPT LOW_DECRYPT 11.39 31.05.99
 \ für Plus-Minus-System

```
\\ Plus-Minus-System

: LOW_ENCRYPT \ plain_char key_char -- cipher_char
+ DUP CHARS >= IF CHARS - THEN ;

: LOW_DECRYPT \ cipher_char key_char -- plain_char
- DUP 0< IF CHARS + THEN ;
```

Screen 3
 \ LOW_ENCRYPT LOW_DECRYPT 20.12 30.05.99
 \ für Minus-Plus-System

```
\\ Minus-Plus-System

: LOW_ENCRYPT \ plain_char key_char -- cipher_char
- DUP 0< IF CHARS + THEN ;

: LOW_DECRYPT \ cipher_char key_char -- plain_char
+ DUP CHARS >= IF CHARS - THEN ;
```

Screen 4
 \ LOW_ENCRYPT LOW_DECRYPT 11.39 31.05.99
 \ für XOR-System

```
\ XOR-System

: LOW_ENCRYPT \ plain_char key_char -- cipher_char
XOR ;

: LOW_DECRYPT \ cipher_char key_char -- plain_char
XOR ;
```

Screen 5
 \ Verschiedene Worte 12.03 30.05.99
 : #? \ d -- new_d \ für höherwertige Stellen in

```
\ <# ... #>
2DUP D0= IF BL HOLD ELSE # THEN ;

: #_ \ d -- new_d \ für höherwertige Stellen in
\ <# ... #>
2DUP D0= IF ELSE # THEN ;

: QI \ --
QUERY INTERPRET ;

: ROVER \ a b c -- a b c a \ "rot over"
2 PICK ;

: ZERO \ adr -- \ Setzt das Wort an der Stelle
\ adr auf 0
0 SWAP ! ;
```

Screen 6
 \ Verschiedene Worte 13.23 31.05.99

```
: U>= \ a b -- flag
U< 0= ;

: INC \ adr -- \ Wert erhöhen
1 SWAP +! ;

: P_ALLOT \ -- \ Weiter. HERE an 16-Bit-Grenze
\ ausrichten
HERE 16 MOD ?DUP IF 16 SWAP - ALLOT THEN ;

: PCREATE \ allotment -- \name \ CREATE an
\ 16-Bit-Grenze
P_ALLOT HERE >R ALLOT R> CONSTANT ;

\ Verwenden Sie PCREATE nicht in Verbindung mit
\ DOES> .
```

Screen 7
 \ CARRAY WARRAY 19.39 30.05.99
 \ Man beachte, daß "base_adr" die von
 \ DOES> gelieferte Adresse bedeutet

```
: CARRAY \ size -- \name \ char array an
\ 16-Byte-Grenze
CREATE HERE >R 0 , P_ALLOT HERE R> ! ALLOT
DOES> \ index base_adr -- adresse
@ + ;

: WARRAY \ size -- \name \ word array
CREATE W* ALLOT
DOES> \ index base_adr -- adresse
SWAP W* + ;
```

Screen 8
 \ 2CARRAY WITHIN 19.39 30.05.99
 \ Man beachte, daß "base_adr" die von
 \ DOES> gelieferte Adresse bedeutet

```
: 2CARRAY \ horz_size vert_size -- \ name
\ 2D char array
CREATE OVER , DUP , * ALLOT
DOES> \ horz_index vert_index
\ base_adr -- adresse
DUP W+ W+ >R \ return: data_adr --
@ \ horz_index vert_index
\ horz_size --
* + R> + ;

: WITHIN \ char niedrigst höchst -- flag
>R >R
DUP R> >= SWAP R> <= AND ;
```

Screen 9
 \ PRINTABLE NUMERIC GERMAN 13.23 31.05.99

```
: PRINTABLE \ char -- flag
32 127 WITHIN ;

: NUMERIC \ char -- flag
ASCII 0 ASCII 9 WITHIN ;

: GERMAN \ char -- flag
\ Umlaute (ä,ö,ü,Ä,Ö,Ü) und ß
>R R@ 132 = R@ 148 = OR
R@ 129 = OR R@ 142 = OR R@ 153 = OR
R@ 154 = OR R> 225 = OR ;
```

\ Das sind char_kind-Filterworte.



```
Screen 10
\ UPPERCASE ALPHA ALPHANUMERIC PUNCTUATION 13.23
31.05.99
: UPPERCASE \ char -- flag
  ASCII A ASCII Z WITHIN ;

: LOWERCASE \ char -- flag
  ASCII a ASCII z WITHIN ;
: ALPHA \ char -- flag
  DUP UPPERCASE SWAP LOWERCASE OR ;

: ALPHANUMERIC \ char -- flag
  DUP ALPHA SWAP NUMERIC OR ;

: PUNCTUATION \ char -- flag
  \ Enthält auch Zwischenraum
  DUP ALPHANUMERIC 0= SWAP PRINTABLE AND ;
\ Das sind char_kind-Filterworte.
```

```
Screen 11
\ Konstanten und Variablen 20.36 30.05.99
  100 CONSTANT KEY_SIZE
KEY_SIZE CARRAY KEY_STRING
KEY_SIZE WARRAY KEY_LENGTHS
  VARIABLE BIG_KEY_LENGTHS
KEY_SIZE CHARS 2CARRAY KEY_CHAR
VARIABLE KEY_LENGTH \ Tatsächliche Schlüssellänge

FILE_SIZE PCREATE CIPHERTEXT
FILE_SIZE PCREATE PLAINTEXT
VARIABLE FILE_MORE \ Dorthin mehr von der Datei
VARIABLE FILE_LENGTH \ Tatsächliche Dateilänge
VARIABLE PAST_CIPHER \ Stelle hinter den gültigen
  \ Daten in CIPHERTEXT

250 CONSTANT NON_CHAR
  \ Für nichtdarstellbare Zeichen
16 CONSTANT DUMP_WIDTH
  \ Zeichenzahl in DUMP horizontal
18 CONSTANT SHOW_KEYS
  \ Von SHOW_KEY gezeigte Zeichen
```

```
Screen 12
\ Konstanten und Variablen DOSINT FILE1
\ 12.06 30.05.99
  300 CONSTANT MAX_SEARCH_SIZE
MAX_SEARCH_SIZE WARRAY COINCIDENCES
10000 CONSTANT UNITY \ Multiplikator für Prozente
\ Prozentzahlen mit Stellen rechts vom Dezimalpunkt

VARIABLE THRESHOLD
\ Schwelle für Wertung als Spitze
CHARS WARRAY FREQS
\ Zahl der Verschlüsselungsergebnisse

VARIABLE 'LOW_ENCRYPT \ Zeiger auf
  \ LOW_ENCRYPT oder LOW_DECRYPT
VARIABLE 'CHAR_KIND \ Zeiger auf char_kind-Prüfwort

DOSINT
0 CONSTANT READ_ONLY
1 CONSTANT WRITE_ONLY
2 CONSTANT READ_WRITE
HCB FILE1 \ Handle für Control-Block
```

```
Screen 13
\ <ENCRYPT> 11.05 27.05.99

VARIABLE SRC \ entweder CIPHERTEXT oder PLAINTEXT
VARIABLE DST \ entweder CIPHERTEXT oder PLAINTEXT

: ADVANCE_KEY_INDEX \ key_index -- new_key_index
  1+ DUP KEY_LENGTH @ = IF DROP 0 THEN ;

: <ENCRYPT> \ quelle ziel --
  \ CIPHERTEXT oder PLAINTEXT
```

```
DST ! SRC ! 0 \ key_index --
FILE_LENGTH @ 0 DO
  SRC @ I + C@ OVER KEY_STRING C@
  'LOW_ENCRYPT PERFORM
  DST @ I + C!
  ADVANCE_KEY_INDEX LOOP
DROP ;
```

```
Screen 14
\ ENCRYPT DECRYPT KEY 20.14 30.05.99

: ENCRYPT \ --
  ['] LOW_ENCRYPT 'LOW_ENCRYPT !
  CIPHERTEXT FILE_SIZE ERASE
  PLAINTEXT CIPHERTEXT <ENCRYPT> ;

: DECRYPT \ --
  ['] LOW_DECRYPT 'LOW_ENCRYPT !
  PLAINTEXT FILE_SIZE ERASE
  CIPHERTEXT PLAINTEXT <ENCRYPT> ;

: GET_KEY \ cipher_char plain_char -- key_char
  LOW_DECRYPT ;
```

```
Screen 15
\ KEY_LENGTH! KEY_STRING! SHOW_KEY_STRING
\ 20.14 30.05.99

: KEY_LENGTH! \ key_length --
  DUP KEY_SIZE >
  ABORT" Für einen Schlüssel zu lang"
  KEY_LENGTH ! ;

: KEY_STRING! \ counted_string --
  COUNT DUP KEY_LENGTH!
  0 DO
    DUP C@ I KEY_STRING C!
    1+ LOOP
  DROP ;

: SHOW_KEY_STRING \ --
  0 KEY_STRING KEY_LENGTH @ DUMP ;
```

```
Screen 16
\ SHOW_PLAIN INIT_KEY_LENGTHS 12.04 30.05.99

: <SHOW_PLAIN> \ von --
  DECRYPT
  PLAINTEXT + 320 DUMP ;
  \ Ungefähr Bildschirmvoll

: SHOW_PLAIN \ --
  0 <SHOW_PLAIN> ;

: INIT_KEY_LENGTHS \ --
  \ Setzt auch BIG_KEY_LENGTHS
  KEY_SIZE 0 DO I KEY_LENGTHS ZERO LOOP
  BIG_KEY_LENGTHS ZERO ;
```

```
Screen 17
\ INPUT_FILE OUTPUT_FILE 20.40 30.05.99

: INPUT_FILE \ filename buffer_ptr --
  >R FILE1 NAME>HCB R@ FILE_SIZE ERASE
  FILE1 READ_ONLY FOPEN
  ABORT" Öffnen der Datei zum Lesen nicht möglich."
  FILE1 R> FILE_SIZE FREAD FILE_LENGTH !
  FILE1 FILE_MORE 1 FREAD
  ABORT" Datei zum Einladen zu groß."
  FILE1 FCLOSE
  ABORT" Schließen der Datei zum Lesen nicht
  möglich." ;
```



```
: OUTPUT_FILE \ filename buffer_ptr --
>R FILE1 NAME>HCB
FILE1 WRITE_ONLY FMAKE
ABORT" Öffnen der Datei zum Speichern nicht
möglich."
FILE1 R> FILE_LENGTH @ FWRITE
FILE_LENGTH @ < ABORT" Diskette ist voll."
FILE1 FCLOSE
ABORT" Datei schließen zum Speichern nicht
möglich." ;
```

```
Screen 18
\ COUNT_COINCIDENCES FILL_PAST_CIPHER 12.07 30.05.99
VARIABLE COIN_COUNT
VARIABLE COIN_SUM
```

```
: COUNT_COINCIDENCES \ cipher_ptr1 cipher_ptr2 -- %
COIN_COUNT ZERO COIN_SUM ZERO
BEGIN DUP PAST_CIPHER @ U< WHILE
OVER C@ OVER C@ = IF COIN_SUM INC THEN
SWAP 1+ SWAP 1+ COIN_COUNT INC REPEAT
2DROP
COIN_SUM @ UNITY COIN_COUNT @ */ ;
\ cipher_ptr1 ist < cipher_ptr2
```

```
: FILL_PAST_CIPHER \ --
CIPHERTEXT FILE_LENGTH @ + PAST_CIPHER ! ;
```

```
Screen 19
\ SEARCH_SIZE KEY_SEARCH_SIZE
\ FILL_COINCIDENCES 12.08 30.05.99
```

```
: SEARCH_SIZE \ -- search_size
FILE_LENGTH @ 3 / MAX_SEARCH_SIZE MIN ;
\ Wir verschieben niemals mehr als ein Drittel der
\ Dateilänge. Dieser Wert wird empirisch bestimmt.
```

```
: KEY_SEARCH_SIZE \ -- key_search_size
SEARCH_SIZE KEY_SIZE MIN ;
```

```
: FILL_COINCIDENCES \ --
\ Übereinstimmungen in CIPHERTEXT
FILL_PAST_CIPHER
SEARCH_SIZE 1 DO
\ minimale Schlüssellänge ist 1
CIPHERTEXT DUP I + COUNT_COINCIDENCES
I COINCIDENCES ! LOOP ;
```

```
Screen 20
\ SHOW_INDEX SHOW_PERCENTAGE
\ SHOW_TABLE_ENTRY 10.47 28.05.99
```

```
: SHOW_INDEX \ index --
0 <# # #? #? #> TYPE ." )" ;
```

```
: SHOW_PERCENTAGE \ percentage --
\ 2 Nachkommastellen
10 / \ Niederwertige Stelle unterdrücken
0 <# # ASCII . HOLD # #? #_ #> TYPE ." " ;
```

```
: SHOW_TABLE_ENTRY \ percentage index --
SHOW_INDEX SHOW_PERCENTAGE ;
```

```
VARIABLE SHOW_FROM
\ Anfangsindex bei den Prozentangaben
48 CONSTANT SHOW_TOTAL
\ Gesamtzahl der Prozentangaben
8 CONSTANT SHOW_ROW
\ Als Nenner in SHOW_TOTAL gedacht
```

```
Screen 21
\ SHOW_COINCIDENCES SHOW_KEY_LENGTHS
\ 10.48 28.05.99
```

```
: SHOW_COINCIDENCES \ from --
\ Zeigt SHOW_TOTAL an FROM
SHOW_FROM ! CR
SHOW_FROM @ SHOW_TOTAL + SEARCH_SIZE
MIN SHOW_FROM @
?DO
I COINCIDENCES @ I SHOW_TABLE_ENTRY
I 1+ SHOW_FROM @ - SHOW_ROW MOD 0= IF CR
THEN
LOOP ;
```

```
: SHOW_KEY_LENGTHS \ -- \ Alle aufzeigen
CR KEY_SIZE 1 DO
I KEY_LENGTHS @ I SHOW_TABLE_ENTRY
I SHOW_ROW MOD 0= IF CR THEN
LOOP
CR ." zu groß = " BIG_KEY_LENGTHS @
SHOW_PERCENTAGE ;
```

```
Screen 22
\ CALC_THRESHOLD 20.14 30.05.99
```

```
VARIABLE COIN_MIN \ kleinster in COINCIDENCES
\ gefundener Wert
VARIABLE COIN_MAX \ größter in COINCIDENCES
\ gefundener Wert
```

```
: CALC_THRESHOLD \ -- schwellwert
\ Mittelwert von COINCIDENCES
100 COIN_MIN ! 0 COIN_MAX !
SEARCH_SIZE 1 DO I COINCIDENCES @
DUP COIN_MIN @ < IF DUP COIN_MIN ! THEN
DUP COIN_MAX @ > IF DUP COIN_MAX ! THEN
DROP LOOP
COIN_MAX @ COIN_MIN @ - 2/ COIN_MIN @ + ;
```

```
Screen 23
\ FILL_KEY_LENGTHS 12.21 29.05.99
```

```
: <FILL_KEY_LENGTHS> \ distance_from_last_spike --
DUP KEY_SIZE < IF \ within key
KEY_LENGTHS INC
ELSE
DROP BIG_KEY_LENGTHS INC THEN ;
```

```
: FILL_KEY_LENGTHS \ -- spike_count
0 0 \ spike_count last_spike --
SEARCH_SIZE 1 DO
I COINCIDENCES @ THRESHOLD @
U> IF \ Eine Spitze gefunden
I SWAP - <FILL_KEY_LENGTHS>
1+ I THEN \ spike_count last_spike --
LOOP
0= ABORT" Es wurde keine Spitze gefunden!" ;
```

```
Screen 24
\ KEY_LENGTHS% CALC_KEY_LENGTH 21.36 30.05.99
```

```
: KEY_LENGTHS% \ spike_count --
\ In Prozent angeben
KEY_SIZE 1 DO
I KEY_LENGTHS @ UNITY ROVER */
I KEY_LENGTHS !
LOOP
BIG_KEY_LENGTHS @ UNITY ROT */
BIG_KEY_LENGTHS ! ;

: CALC_KEY_LENGTH \ -- length
INIT_KEY_LENGTHS FILL_KEY_LENGTHS KEY_LENGTHS%
0 \ max_key_length --
KEY_SIZE 1 DO
I KEY_LENGTHS @ OVER KEY_LENGTHS @ > IF
DROP I THEN
LOOP ;
```



\ CALC_KEY_LENGTH verwendet den kleineren Index,
 \ wenn zwei zu Elementen mit demselben Wert gehören.

Screen 25
 \ FILL_KEY_LENGTH TRY 20.10 30.05.99

```
: FILL_KEY_LENGTH \ --
  FILL_COINCIDENCES 1 SHOW_COINCIDENCES
  CALC_THRESHOLD THRESHOLD !
  CR ." Schwellwert = " THRESHOLD @
  SHOW_PERCENTAGE
  CALC_KEY_LENGTH KEY_LENGTH! SHOW_KEY_LENGTHS
  CR ." Schlüssellänge ist: " KEY_LENGTH @ . ;

: TRY \ plain_char horz_index vert_index --
  16 * + >R R@ KEY_LENGTH @ MOD KEY_STRING
  R> CIPHERTEXT + C@
  \ plain_char key_ptr cipher_char --
  ROT GET_KEY SWAP C!
  SHOW_PLAIN ;

\ TRY geht davon aus, daß PLAINTEXT an 16-Bit-Grenze
\ ausgerichtet ist. TRY faßt PLAINTEXT als 2D-Array
\ mit 16 Eingängen auf (wie DUMP zeigt).
```

Screen 26
 \ INIT_FREQS FILL_FREQS 12.33 30.05.99

```
: INIT_FREQS \ --
  CHARS 0 DO I FREQS ZERO LOOP ;

: FILL_FREQS \ cipher_ptr -- \ Schrittgröße =
KEY_LENGTH
  INIT_FREQS
  PAST_CIPHER @ SWAP DO
  I C@ FREQS INC
  KEY_LENGTH @ +LOOP ;
```

Screen 27
 \ BEST_CIPHER_CHAR SINGLE_FILL_KEY 20.10 30.05.99

```
: BEST_CIPHER_CHAR \ -- best_cipher_character
-1 -1
\ best_cipher_char best_cipher_char_occurrences --
CHARS 0 DO \ I ist das Prüfzeichen
  I FREQS @ OVER > IF 2DROP
  I I FREQS @ THEN
  LOOP
-1 = ABORT" FREQS fehlerhaft" ;

: SINGLE_FILL_KEY
\ horz_index vert_index -- best_cipher_char
KEY_CHAR >R BEST_CIPHER_CHAR
DUP MOSTEST C@ GET_KEY R> C! ;
```

Screen 28
 \ COLUMN_FILL_KEY FILL_KEY 12.34 30.05.99

```
: COLUMN_FILL_KEY \ horz_index --
  CHARS 0 DO \ I ist der vert_index
  DUP I SINGLE_FILL_KEY
  \ horz_index best_cipher_char --
  FREQS -1 SWAP !
  \ Ist nicht der beste des nächsten
  \ vert_index
  LOOP
  DROP ;

: FILL_KEY \ --
  FILL_PAST_CIPHER
  KEY_LENGTH @ 0 DO \ I ist horz_index
  CIPHERTEXT I + FILL_FREQS
  I COLUMN_FILL_KEY
  LOOP ;
```

Screen 29
 \ SHOW_KEY SHOW_KEY_HEX 19.18 29.05.99

```
: SHOW_KEY \ --
  CR
  SHOW_KEYS 0 DO \ J = vert_index
  KEY_LENGTH @ 0 DO \ I = horz_index
  I J KEY_CHAR C@ DUP PRINTABLE IF
  EMIT ELSE DROP NON_CHAR EMIT THEN
  SPACE LOOP CR LOOP ;

: SHOW_KEY_HEX \ --
  CR BASE @ >R HEX
  SHOW_KEYS 0 DO \ J = vert_index
  KEY_LENGTH @ 0 DO \ I = horz_index
  I J KEY_CHAR C@ 0 <# # # BL HOLD #> TYPE
  LOOP CR LOOP
  R> BASE ! ;
```

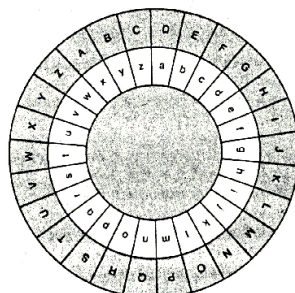
Screen 30
 \ TO_KEY_STRING FILL_KEY_STRING 19.47 30.05.99

```
: <TO_KEY_STRING> \ --
  KEY_LENGTH @ 0 DO \ J = horz_index
  0 I KEY_STRING C! \ Vorgabe
  CHARS 0 DO \ I = vert_index
  J I KEY_CHAR C@ DUP 'CHAR_KIND PERFORM
  IF
  J KEY_STRING C! LEAVE ELSE DROP
  THEN
  LOOP
  LOOP ;

: TO_KEY_STRING \ char_kind_cfa --
  'CHAR_KIND ! <TO_KEY_STRING> ;


: FILL_KEY_STRING \ --
  FILL_KEY SHOW_KEY
  [' ] ALPHA TO_KEY_STRING SHOW_KEY_STRING ;
```

Aus dem Archiv:



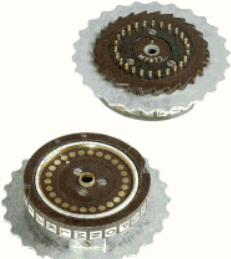
Chiffrierscheibe für eine Kodierung auf der Basis eines Verschiebealgorithmus‘.

Rotormaschine M 94, United States, polyalphabetische Substitution.



ENIGMA

Rotorscheiben,
polyalphabetische Substitution





Spaß mit Forth

**Hugh Aguilar, Chris Jakeman,
Willem Ouwerkerk, Friederich Prinz,
Martin Bitter, Fred Behringer**

Die vorliegenden Beiträge aus vier Ländern wurden von einigen Anmerkungen in einem Artikel der Forth Dimensions von Hugh Aguilar über das Knacken von Geheimcodes angeregt. In den vergangenen Jahren wurden im Zusammenhang mit den ANSI-Standards und der `comp.lang.forth` sehr stark die Vorteile hervorgehoben, die professionelle Programmierer aus der Verwendung von Forth ziehen können. Die Autoren des vorliegenden Artikels begrüßen diese Vorteile, wollen aber daran erinnern, daß Forth gerade auch für den Amateur-Forthler geeignet ist und daß eben gerade Forth in vieler Hinsicht geeigneter ist als andere, komplexere Sprachen.

Hugh Aguilar schreibt in seinem Forth-Dimensions-Artikel über das Code-Knacken: "Uns machte schon das Schreiben von CrakPoly viel Spaß und wir fanden, daß uns der Umgang mit diesem Programm ebensoviel Spaß bereitete. Außerdem ist das Schreiben und der Entwurf eines Programms, das keinen unmittelbaren Zweck erfüllen soll, auch für das Arbeiten an kommerziellen Produkten eine gute Vorbereitung.

C++ mit seiner Betonung von GUIs und kommerzieller Entwicklung erfordert für ein Wochenend-Projekt einen viel zu großen Aufwand. Da es niemanden mehr gibt, der das Programmieren nur so aus Spaß an der Freude betreibt und dabei seine Programmiererfahrung erweitert, wird unsere professionelle Programmierung heutzutage mit Ausdrücken wie "Gewaltmarschprojekt" und "gegen jede Regel" belegt. Das scheint der Lohn der Professionalität zu sein."

Dieser Artikel wird in englischer Version auch in der Forth Dimensions (FIG USA) und in der Forthwrite (FIG UK) veröffentlicht. Eine holländische Version wird in 'Het Vijgeblaadje' (HCC-Forth-gebruikersgroep) erscheinen.

Warum "Forth aus Vergnügen am Programmieren"?

1. Programmieren nur so zum Spaß macht Spaß. Grund genug, es zu tun.
2. Programmieren, ohne einen direkten Zweck zu verfolgen, macht noch größeren Spaß, wenn sich andere daran beteiligen. Das Programm gewinnt dadurch an Nützlichkeit und wird interessanter.

3. Forth Dimensions und deren Schwesterzeitschriften sind praktisch die einzigen heute noch übriggebliebenen Zeitschriften, die Artikel über zweckungebundene Programme veröffentlichen und deren Quelltexte bereitstellen. Das ist Grund genug, in Forth zu programmieren.
4. Forth ist in idealer Weise dazu geeignet, Programme nur so zum Zeitvertreib zu schreiben, da es sehr einfach aufgebaut ist.
- 4a. Portierbares Forth stellt eine Kommandozeilen-Schnittstelle bereit, so daß der Anwendungsprogrammierer kein GUI-Interface zu schreiben braucht. Forth ist robust, aber auch einfach, und mit Leichtigkeit können Erweiterungen wie OOP und GUI angefügt werden.
5. Wenn man spielerische Programme schreibt, wird man zum besseren Programmierer. Das hat auch auf das Berufsleben (Forth oder sonstwie) einen positiven Einfluß.
6. Programme, die nur so zum Spaß geschrieben werden und anfangs keinen kommerziellen Wert zu haben scheinen, können sich zu einem wirklich guten und wertvollen Produkt entwickeln. Wenn eine größere Zahl von Benutzern in das Programm einsteigt, kann der Autor sogar daran

(Englische Forth-Gesellschaft)

Treten Sie unserer Forth-Gruppe bei.
Verschaffen Sie sich Zugang zu unserer umfangreichen Bibliothek.
Sichern Sie sich alle zwei Monate ein Heft unserer Vereinszeitschrift.
(Auch ältere Hefte erhältlich)
Suchen Sie unsere Webseite auf:
www.users.zetnet.co.uk/aborigine/Forth.htm
Lassen Sie sich unser Neuzugangs-Gratis-Paket geben.
Der Mitgliedsbeitrag beträgt 12 engl. Pfund.
Hierfür bekommen Sie 6 Hefte unserer Vereinszeitschrift Forthwrite.
Beschleunigte Zustellung (Air Mail)
ins Ausland kostet 20 Pfund.
Körperschaften zahlen 36 Pfund,
erhalten dafür aber viel Werbung.

Wenden Sie sich an:

Dr. Douglas Neale
58 Woodland Way
Morden Surrey
SM4 4DS
Tel.: (44) 181-542-2747
E-Mail: dneale@w58wmorden.demon.co.uk



denken, mit Dienstleistungen und benutzerspezifischen Zusätzen zu Geld zu kommen. Genau das, nämlich aus der Public-Domain heraus und mit offen dargelegten Quelltexten Geld zu verdienen, war die Idee von Richard Stallman.

Es gibt da sicher eine Menge ansprechender Forth-Programme, die im Dunkeln dahinschlummern, weil der Autor meint, daß sich niemand für sein Programm interessiert und es keinen kommerziellen Wert hat. Wir müssen diesen unbekannteren Forth-Programmierer dazu ermutigen, einen Artikel zu schreiben und ihn zu veröffentlichen. Die Leute warten nur darauf, ihn zu lesen.

Hugh Aguilar, FIG International, Autor zahlreicher Artikel in der Forth Dimensions

Forth für zu Hause, aber nicht für die Arbeit - wozu sich aufregen?

Ich versuche mich nun schon seit 15 Jahren an Forth und bin seit fast genauso vielen Jahren Mitglied der englischen Forth Interest Group (FIG UK). Ich habe nie eine Zeile Forth berufsmäßig geschrieben und mein Einsatz von Forth im Berufsleben beschränkte sich auf Einzeiler wie

```
50 RANDOM 1+ .
```

mit dem ich den Gewinner in einer Werbelotterie mit 50 Teilnehmern ermittelte.

In diesen 15 Jahren habe ich kleine Software-Arbeitsgruppen geleitet, die schwierige Aufgaben in C, C++, Unix und Windows erledigten. Forth war ein wichtiger Bestandteil meiner eigenen ständigen Weiterentwicklung.

Blättert man die Seiten der zurückliegenden Hefte der Forthwrite, der Vereinszeitschrift der FIG UK, durch, so wird man sehen, daß sie gespickt sind mit den verschiedensten Arbeitsberichten von mir über die Ergebnisse von Experimenten in Sprachentwurf (Parser, Compiler, objekt-orientiertes Programmieren), Stringsuche, Mustererkennung, Speicherverwaltung usw.

Das alles wurde nur so "aus Spaß", aus einem gewissen Forschungsdrang heraus, auf die Beine gestellt und lieferte ein paar neue Algorithmen (darunter ein sehr schnelles Suchverfahren für Zeichenketten ohne Groß/Klein-Unterscheidung, nach welchem ich in den Büchern vergeblich gesucht hatte). "Aus Spaß", aber es half mir sehr dabei, meinen Mitarbeitern eine Nasenlänge voraus zu bleiben. Sie kommen zu mir, wenn ihnen die Ideen ausgehen (wirklich höchst erfreulich).

Wenn Sie mich fragen, ob diese ganzen Untersuchungen denn nicht auch in einer gängigeren Sprache vorgenommen werden konnten, würde ich antworten: "Ja, aber nur in Forth ist die Arbeit kurz genug, um in einen Artikel zu passen." Aus Prinzip, weil Forth klein, einfach und kinderleicht (Sie werden

vielleicht sagen "primitiv") aufgebaut ist, hat es mich gedrängt, Binärsuchverfahren oder Speicherverwaltung näher zu untersuchen. Auf einem Unix-Computer sind diese Dinge einfach vorhanden und werden als selbstverständlich betrachtet.

Mir hat Forth immer Spaß gemacht. In meinem Beruf, der professionellen Seite meines Lebens, hat es mich nur am Rande berührt, war aber nichtsdestoweniger ständig da. Ich werde weiterhin an Forth Spaß haben und ich werde mich weiterhin über Kontakte zu anderen Forthlern freuen. Sie sind durch die Bank weg interessantere Leute als die meisten anderen.

Chris Jakeman, Redakteur der Forthwrite, FIG UK

Spielen und Lernen

Ich hatte schon immer viel Spaß daran, mir meine eigenen Spielsachen und Werkzeuge selbst zu basteln. Als Kind und als Heranwachsender habe ich mich mit Modellbau beschäftigt. Ich entwarf und baute Boote, Rennwagen und anderes mehr. Heute sind es die Forth-Systeme, meine eigenen und die anderer, die mich in Verbindung mit einer gehörigen Portion Kenntnis in Elektronik in die Lage versetzen, mir mein eigenes Spielzeug zu entwerfen. Auch mein Interesse am Verhalten von Mensch und Tier kann ich dadurch zur Geltung bringen: Mit Forth entwickle ich Maschinen (Roboter), die von einem in Software gegossenen Verhaltenstypus gesteuert werden. Hier gehe ich genauso vor: Forth gestattet es mir, die Probleme intuitiv (experimentierend) zu ergründen. Diese Roboter sind an sich nutzlos, aber sie mir auszudenken und sie auszuführen, bereitet mir größte Genugtuung. Außerdem ist der Begriff "nutzlos" hier sicher zu eng gefaßt. Allein das Lächeln und das Staunen der Besucher einer Ausstellung machen es schon die Mühe wert.

Schon seit Jahren versuche ich mit wechselndem Erfolg, diesen Enthusiasmus weiterzugeben, mit Artikeln im *Vijgeblaadje*, mit Anleitungen und mit Büchern, die ich über den niederländischen Forth-Club verbreite. Das *Igel-Arbeitsbuch* entsprang dieser Absicht. Auch in meinem Berufsleben versuche ich stets, mir die Freude an der Arbeit zu bewahren. Ein elegant entworfenes Stück Soft- oder Hardware erfüllt mich noch immer mit tiefer Zufriedenheit, und am allermeisten dann, wenn es mit einem Minimum an Mitteln und Aufwand erreicht werden konnte.

Ein Großteil der Arbeit bei der Entwicklung einer neuen Forth-Implementation besteht daraus, das Rad neu zu erfinden. Weite Teile von Forth sind wohlbekannt, aber die optimale Ausnutzung der gegebenen Möglichkeiten ist eine Kunst für sich. Ich schöpfe daher auch tiefe Zufriedenheit aus dem Erkunden eines Prozessors, um ihn dann mit selbstgeschriebener Software (ByteForth) so einzurichten, wie ich ihn brauche. Meiner Meinung nach sind professionelle Systeme nicht immer die bessere Wahl, wenn es auch unter den Software-Machern ein paar geben mag, die schlauer sind als ich.



Eigenartigerweise sind viele Selbstbau-Forth-Implementationen nur mit unzureichender Dokumentation ausgestattet. Ich selbst versehe jede Implementation auch immer gern mit einer ausgedehnten Dokumentation, - die ich dann auch regelmäßig verwende! Vielleicht liegt das daran, daß ich meine eigenen Systeme auch zur Schaffung von Robotern und zur Ausübung verschiedener Arbeiten für andere benutze. Ein gutgeschriebenes Handbuch ist eine Augenweide und macht das (nicht-)professionelle System auch für andere brauchbar.

Willem Ouwerkerk, Redakteur des Vijgeblaadjes und Vorsitzender der HCC-Forth-gebruikersgroep

Forther helfen Forthern

Jeder Forther hat sich selbst und anderen Forthern schon die Frage gestellt, warum man sich ausgerechnet mit dieser Sprache und mit einer virtuellen Maschine aus den frühen 70ern beschäftigt. Die Antworten kennen Sie.

Wenn das Gespräch zeitlich über den gegenseitigen Austausch der wohlbekannten Argumente hinaus gelangt, kommt aber immer auch ein Aspekt zur Sprache, den ich ganz persönlich im Zusammenhang mit Forth besonders schätze. Dieser „Aspekt“ sind die Menschen, die sich mit Forth auseinandersetzen.

Diese Menschen nutzen den Computer zur Lösung von Aufgaben. Oft sind das nicht die eigenen Aufgaben, sondern Aufgaben, die durch andere - meist als Hilfesuche - an sie herangetragen werden. Um Hilfe gebeten, vergessen die meisten Forther nur zu gerne die Zeit und ihre eigenen Interessen, und verbringen Tage und Nächte mit der Lösung eines Problems, das nicht ihr eigenes ist. Dabei produzieren sie manchmal kleine Werkzeuge für andere, und manchmal große und komplexe Forthsysteme. Und fast immer geben sie die Ergebnisse ihrer Arbeiten großzügig und freigiebig an jedermann weiter, der diese Ergebnisse für sich und seine Aufgaben verwerten kann.

Mir scheint, daß die Forther in besonderer Weise daran gewöhnt sind, in ganz kleinen Schritten zu denken und zu handeln. Vernetztes, übergreifendes Denken und die gerade in der Informatik viel gepriesene Arbeitsteilung sind den Forthern nicht fremd. Aber sie wollen sich die Verantwortung auch für die kleinen Dinge der großen Probleme und komplexen Aufgaben nicht nehmen lassen. Forther fühlen sich verantwortlich für die Kerne in Fragestellungen, von denen sie sich berührt sehen.

Und dabei habe ich nur wenige Forther erlebt, die sich selbst als zu groß für triviale Aufgaben betrachtet haben. Solche Forther gibt es natürlich auch. Aber die erscheinen mir als zu klein, um sie in große Aufgaben einzubeziehen. Und unsere größte Aufgabe für Forth und für einander ist nach wie vor, das, was uns Freude an Forth bereitet, möglichst vielen anderen Menschen nahezubringen.

Forth macht Spaß. Mindestens genau so großen Spaß machen mir die Forther !

Friederich Prinz, Direktor der Forth-Gesellschaft und Redakteur der Vierten Dimension, Swap-Drachen-Preisträger 1995

Forth - oder die Kunst, ein Motorrad zu pflegen

Jeden Arbeitstag fahre ich mit meinem Auto zur Schule. Und ich muss sagen, dass ich (Auto)Fahren überhaupt nicht mag. Vielmehr genieße ich es, mein altes Motorrad (fast genauso alt wie ich) über kurvige, sanftgeschwungene Landstraßen zu fahren, welch Gefühl, mit leicht (sic!) durchdrehendem Hinterrad aus einer Kurve heraus zu beschleunigen! (Für die, die es näher interessiert: DKW RT 175 S, Bj: Okt. 1953 9 PS). Als ich die DKW vor über 15 Jahren bekam, war sie unvollständig und in drei großen Umzugskartons verpackt. Wie ich inzwischen weiß, lautet eine Restauratorenregel: "Kaufe nie ein zerlegtes Motorrad!" Glücklicherweise kannte ich diese Regel damals nicht!

Nichtsdestotrotz, ich brachte die Maschine ans Laufen und kenne jetzt jede Schraube in und an ihr. Wirklich jede! Und alle ihre Funktionen. Aber ich bin niemals zu einem DKW-Experten geworden, der auf einen Blick Baujahr und Modell erkennt, die Firmengeschichte wiedergeben kann und die Namen der Konstrukteure weiß.

Es macht(e) Spaß, an so einer Maschine zu schrauben, und es macht Spaß, diese Maschine zu fahren. Trotzdem fahre ich zur Arbeit mit dem Auto - es ist praktischer, bequemer und (schwer zu sagen) zuverlässiger. Genau so wenig schraube oder bastle ich an dem Auto herum - das überlasse ich der Werkstatt meines Vertrauens.

Ähnlich ist es mit Forth: Ich benutze mit viel Vergnügen das von Tom Zimmer vor langer Zeit entwickelte ZF, das jetzt von der Forth-Gruppe Moers gepflegt wird. Ich kenne zwar nicht jedes Byte dieses ZFs, aber wenn ich es ernsthaft versuchte: Es wäre möglich! Die Strukturen dieses Forthes (und die damaligen Denkwege von Tom Zimmer) zu entschlüsseln, bereitet mir Genugtuung, und alles was ich dazu brauche, ist ZF selbst!

Ich mag die Lernmöglichkeiten, die Forth mir bei der Entschlüsselung seiner Interna bietet, und benutze das erworbene Wissen als ein Sprungbrett, die Geheimnisse von DOS zu enträtseln. Als meine Beziehung zu Forth begann, kannte ich keine einzige „goldene“ Programmierregel oder Standards der Informatik, wie „Trenne immer Code und Daten! ;-)“. Die einzige Regel hieß: Rechner läuft = gut; Rechner hängt = schlecht! Forth ist also letztendlich ein Vergrößerungsglas, mit dem ich die Tiefen verschiedener OSe erkunden kann, und es ist selbst durchsichtig bis ins letzte Byte.

Natürlich hat das Lesen einiger Bücher weitergeholfen - unter



den ersten waren die beiden Brodie-Klassiker. Viel mehr aber noch hat das Studium von Quelltexten oder Auszügen und Codeschnipseln anderer Forthprogrammierer geholfen, die in der „Vierte Dimension“ veröffentlicht wurden. Programmierer unterschiedlicher Couleur zeigten dort ihre Gedanken und ihren Code, und ich saugte das auf wie ein Schwamm.

Die Forthe, die ich benutze, sind billig (preiswert sind noch einige andere). Als ambitionierter Hobbyist bin ich nicht in der Lage, viel Geld für eine Programmiersprache auszugeben - andererseits will (muss) ich ja auch kein Geld mit Forth verdienen.

Nur so aus Spaß! Aber manchmal hilft Forth mir, meinen Schülern zu helfen oder Probleme zu lösen, für die ich keine fertigen Programme kaufen kann. Ich habe viel aus den Sourcecodes anderer gelernt, selbst wenn dieser Code nicht immer perfekt war. Und es ist daher meine felsenfeste Überzeugung: **Jeder**, der aus Spass und Vergnügen oder auch, weil er damit sein Geld verdient, Forth-Code schreibt, sollte veröffentlichen! Die Arbeit des 'Polierens' für einen Artikel wird mehr als aufgewogen durch das Gefühl, mit der Forth-Gemeinde etwas zu teilen!

Martin Bitter, Forth-Gesellschaft, Verfasser einer Reihe von Artikeln in der Vierten Dimension, aktiver Mit-Chatter im englischen Kanal #FIGUK

Weiterbildung durch Freizeit-Forth

"Publish or perish" ist ein in Wissenschaftlerkreisen allgemein bekanntes geflügeltes Wort. Es geht also gar nicht so sehr um die Suche nach Wahrheit, es geht ums reine Überleben? Das ist kein Spaß. Das "Ranking" der Universitäten heutzutage und die "Evaluation", die Einstufung und Bewertung, stützen sich zum größten Teil auf eine Aufrechnung von Veröffentlichungen, die im Science Citation Index genannt werden. Das ist die eine Seite des Lebens, die professionelle. Die andere Seite ist der wohlberechtigte Wunsch eines jeden kreativen Menschen, Dinge zu tun, die ihm Spaß machen, ganz egal, ob sinnvoll oder nicht.

Wie andere Forthler auch, bin ich ein Fachmann auf meinem eigenen Gebiet. Ich brauche kein Forth, um mich als Experten auszuweisen. Dazu wäre mir Forth zu eng. Ich brauche Forth zur Entspannung, als Freizeitbeschäftigung mit Themen, die nicht allzu weit von meinem beruflichen Arbeitsgebiet entfernt sind. Ich beschäftige mich gern mit Dingen, die keinen besonderen Zweck verfolgen. Und dazu habe ich mir Forth auserkoren. Das ist kein Zufall. Es gab eine Zeit, da ich nach einer maschinennahen Sprache Ausschau hielt, mit der ich den Digitalteil eines Hybridrechners dazu bringen konnte, den Analogteil automatisch zu warten. ALGOL und FORTRAN waren mir keine Hilfe und Assembler war ermüdend. Also erfand ich meine eigene Sprache, DISPRA (Dialogsystemsprache). Das war 1969. Später erkannte ich, daß Forth DIE Sprache für diesen Zweck hätte sein können. Da war ich dann

Holländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Und außerdem ist Forth sowieso international. Neugierig ? Werden Sie Förderer der

HCC-Forth-gebruikersgroep.

Für 20 Gulden pro Jahr schicken wir Ihnen 5 oder 6 Hefte unserer Vereinszeitschrift 'Het Vijgeblaadje' zu. Dort können Sie sich über die Aktivitäten unserer Mitglieder, über neue Hard- und Softwareprojekte, über Produkte zu günstigen bezugspreisen, über Literatur aus unserer Forth-Bibliothek und vieles mehr aus erster Hand unterrichten. Auskünfte erteilt:

Willem Ouwerkerk
Boulevard Heuvelink 126
NL-6828 KW Arnhem
E-Mail: w.ouwerkerk@kader.hobby.nl

Oder überweisen Sie einfach 20 Gulden auf das Konto 525 35 72 der HCC-Forth-gebruikersgroep bei der Postbank Amsterdam. Noch einfacher ist es wahrscheinlich, sich deshalb direkt an unseren Vorsitzenden, Willem Ouwerkerk zu wenden.

aber schon auf anderen Gebieten der Wissenschaft tätig. Nichtsdestoweniger wurde ich Forth-süchtig. Die professionelle Seite meines Lebens liegt inzwischen hinter mir und ich habe entdeckt, daß Forth auch für den Ruhestand überaus geeignet ist. Ich bleibe aktiv und brauche mich dabei nicht mehr mit dem unmenschlichen Konkurrenzgeplänkel des ernsthaften wissenschaftlichen Forschens herumzuschlagen.

Ich bin der notorische Hobby-Forthler, der "Enthusiast, der die Neigung hat, das Rad immer wieder neu zu erfinden". Mir macht das Wiedererfinden Spaß. Nur durch Wiedererfindung komme ich dazu, die Dinge wirklich zu verstehen. Ich bin auch derjenige, der sich nicht scheut, "Ich-auch-Compiler" zu bauen. Und ich habe mich zum Beispiel nicht darum gekümmert, ob Transputer-Forth irgendeine Zukunft hat, nicht damals und nicht heute. Ich bin neugierig. Mir macht Lernen Spaß. Ich lerne gern durch Ausprobieren. Durch Ausprobieren lernt man Dinge, die man aus den Büchern nie lernen kann. Ich habe überhaupt nichts dagegen, wenn man Forth als eine Art Religion bezeichnet.

Ich betrachte Forth nicht so sehr als Sprache. Für mich ist Forth eine Idee. Ich beteilige mich gern an Linux-artigen Projekten, die von einer Gruppe von Enthusiasten im Internet durchgezogen werden. Es erinnert mich an Arbeiten innerhalb der weltweit organisierten internationalen Gemeinde von Wissenschaftlern und Forschern, nur daß ich mich in Forth auf die mehr unterhaltsame Seite zurückziehen kann. Ich bin dagegen, irgendwelche Gelder für irgendwelche Forth-Systeme auszugeben, mögen sie noch so genial entworfen



Reed-Solomon-Fehlerkorrektur

sein. Mir gefällt die Stallmansche Idee der frei zugänglichen Quellen. In der Wissenschaft ist das seit mehr als zweihundert Jahren gang und gäbe. Man stelle sich einen Mathematiker vor, der einen neuen mathematischen Satz veröffentlicht, ohne den Beweis dazu anzugeben! Ich nehme gern Kontakt zu anderen Forthlern auf, die gern ... siehe oben. Meine Vorliebe gilt Forth.

Fred Behringer, FIG US, FIG UK, Forth-Gesellschaft, Swap-Drachen-Preisträger 1999

Reed-Solomon-Fehlerkorrektur

(Teil 2)

von Glenn Dixon
<Dixong@iomega.com>,
Roy, Utah, USA

Mit freundlicher Genehmigung der
amerikanischen Forth Interest Group
übersetzt von Fred Behringer, München.

Der Originalaufsatz erschien in der Forth Dimensions,
Band XX, Nummer 5,6, vom Januar/April 1999.

Fehlerkorrekturen nach Reed-Solomon sind Vorwärtskorrekturen. Sie werden in Diskettenlaufwerken, CDs, Satelliten und anderen Übertragungskanälen verwendet. Die Daten werden vor dem Absenden redundant erweitert. Am Bestimmungsort läßt sich erkennen, ob ein Fehler vorliegt, und dieser läßt sich korrigieren, ohne daß die Daten erneut übertragen werden müssen.

Stichworte: Fehlererkennung, Fehlerkorrektur, redundanter Code, endlicher Körper, irreduzibles Polynom

Teil 1 (VD 4/99) brachte eine Einführung in die Arithmetik auf endlichen Körpern. Das ist diejenige Art von Mathematik, die in vielen Computer-Algorithmen verwendet wird, z.B. bei der Fehlerkorrektur, der Datenverschlüsselung und der Erzeugung von Zufallszahlen. Ihre Verwendung verdankt sie dem Umstand, daß sie sich höchst effizient sowohl in Hardware als auch in Software implementieren läßt. Im vorliegenden Artikel wird die Reed-Solomon-Fehlerkorrektur in Entwurf und Anwendung diskutiert.

Beim Entwurf eines Reed-Solomon-Korrektors sind verschiedene Parameter zu beachten: Codewortlänge, erzeugendes Polynom und Zahl der redundanten Elemente. Wir wollen hier die Vorgehensweise grob skizzieren.

Reed-Solomon-Fehlerkorrekturcodes (FKCs) bearbeiten die Daten immer nur in ganzen Blöcken, gewöhnlich von gleichbleibender Größe, die vom Entwickler vorgegeben wird. Will man einen kontinuierlichen Datenstrom gegen auftretende Fehler schützen, so muß man diesen in Blöcke oder Portionen

von bestimmter Länge einteilen: Manchmal ergibt sich die zu wählende Blockgröße in ganz natürlicher Weise. Will man zum Beispiel Daten in 128-Byte-Päckchen verschicken, dann nehme man das als Blockgröße. Die zu schützenden Daten werden vom FKC nicht verändert. Es werden lediglich FKC-Elemente an das Blockende angefügt und der neue, größere Block wird in seiner Gesamtheit als Datenblock verschickt. Die maximale Größe eines Blockes, der fehlergeschützt werden kann, bestimmt sich aus der gewählten Codeelementlänge wie folgt:

Maximale Blockgröße = (2 hoch Bitzahl im Codeelement) - Anzahl hinzugefügter FKC-Elemente - 2

Wenn wir zum Beispiel ein 8-Bit-Codeelementwort (Byte) wählen und uns dafür entscheiden, den Datenblock mit 16 Bytes an FKC zu schützen, würde die Länge des maximalen Datenblocks, den wir behandeln können, $256 - 16 - 2$, also 238 Bytes, betragen. Mit kürzeren Blöcken würde es auch gehen. Die Datenelemente müssen von gleicher Größe oder kleiner als die FKC-Elemente sein. Bei byte-orientierten Datenblöcken verwendet man daher häufig 8-Bit-Elemente. Werden größere Blöcke benötigt, kann man *Verzahnungsmethoden* (interleaving techniques), siehe weiter unten, anwenden.

Wieviele Korrektur Elemente den einzelnen Datenblöcken zugefügt werden sollen, hängt vom gewünschten Korrekturvermögen ab. Zur Fehlerbeseitigung bei einem Datenelement benötigt man zwei redundante Elemente (*FKC-Elemente* oder *Korrektur Elemente*). Man kann sich das so vorstellen: 1 Element, um herauszufinden, wo der Fehler steckt, und 1 Element, um herauszufinden, was an dessen Stelle eigentlich stehen sollte. Wenn wir bei einem byte-orientierten Datenblock nur ein einziges fehlerhaftes Byte korrigieren wollten, würden wir an den betreffenden Block zwei Bytes anhängen. Es hat sich eingebürgert, mindestens sechs bis acht Fehlerschutz-Elemente zu verwenden, und wenn man eine gute Fehler-schutzwirkung erzielen möchte, wir sprechen noch darüber, muß man mit dieser Zahl noch höher gehen.

Im obenstehenden Beispiel wurden dem Datenblock sechzehn FKC-Bytes angefügt. Der FKC kann damit bis zu acht fehlerhafte Bytes richtigstellen. Die fehlerhaften Bytes können irgendwo im empfangenen Block stecken, auch in den FKC-Bytes selbst. Das Reed-Solomon-Verfahren ist jedoch so angelegt, daß im empfangenen Datenblock überhaupt nichts mehr korrigiert werden kann, wenn die Zahl der aufgetretenen Fehler die Zahl der korrigierbaren Fehler übersteigt. Wenn also im obigen Beispiel neun Bytes fehlerhaft sind, setzt der Korrekturalgorithmus aus und es kann kein einziges Byte mehr repariert werden.

Falls alle Korrektur Elemente aufgebraucht worden sein sollten, kann man überdies nicht mit Sicherheit sagen, ob das Korrekturvermögen überschritten wurde oder nicht. Dann kann es also vorkommen, daß wir meinen, der FKC habe



sämtliche Fehler korrigiert, obwohl in Wirklichkeit noch mehr Fehler vorhanden waren, die überhaupt nicht korrigiert wurden. Wir können die Wahrscheinlichkeit einer "fehlerhaften Fehlerkorrektur" dadurch reduzieren, daß wir entweder auf einiges an Korrekturvermögen verzichten oder mehr Prüfelemente verwenden oder aber (noch) andere Datensicherungsverfahren einsetzen, wie zum Beispiel CRC (cyclic redundancy check - zyklische Redundanzprüfung). Im FKC-Software-Paket sind einige weitere Anmerkungen hierzu zu finden.

Die Auswahl eines aus den zur Verfügung stehenden irreduziblen Polynomen und des Offsets ist recht subtil und spielt in den meisten Anwendungen keine große Rolle. Werden Teile des Algorithmus in Hardware ausgeführt, kann eine geschickte Wahl zu Einsparungen führen. Ich kann keine spezielle Empfehlung geben. Alle zulässigen Polynome sind gleichgut und führen zum selben Korrekturvermögen und zur selben Wahrscheinlichkeit unzureichender Korrektur über den gesamten Datensatz hinweg.

Zur Erzeugung der FKC-Elemente auf der Sendeseite wird ein *Codierer* (Abbildung 1-a) verwendet. Die FKC-Elemente werden dann mit den ursprünglichen Daten zusammen verschickt. Auf der Empfangsseite wird ein *Decodierer* (Abbildung 1-b) dazu verwendet, empfangene Fehler zu entdecken und jene Informationen bereitzustellen, die nötig sind, um die aufgetretenen Fehler zu korrigieren. Der Codierer ist in Form eines Polynomteilers angelegt, der Prüfelemente erzeugt. Wenn die Prüfelemente an die Datenelemente angehängt werden, ist der entstandene Gesamtdatensatz gleichmäßig durch ein bestimmtes Polynom teilbar, das das *erzeugende Polynom* genannt wird.

Da die Zahl der durch das erzeugende Polynom gleichmäßig teilbaren Datensätze gegenüber der Zahl aller möglichen Datensätze verschwindend gering ist, wird so ziemlich jede Fehlerkombination den Datensatz so verändern, daß er nicht mehr gleichmäßig durch das erzeugende Polynom teilbar ist. Wir können also den empfangenen Datensatz dadurch auf Vorliegen eines Fehlers testen, daß wir ihn durch das erzeugende Polynom teilen und nachsehen, ob ein Rest bleibt. Das macht der Decodierer.

Wenn Fehler festgestellt wurden, enthalten die Register des Decodierer-Blocks *Syndrom*-Elemente. In diesen Elementen sind die Informationen enthalten, die zur Lokalisierung und Korrektur der Fehler nötig sind. Für jeden Fehler im Datensatz werden zwei Parameter benötigt: Die Stelle, an der der Fehler auftrat (gemessen als Zahl der Elemente vom Ende des Datensatzes aus), und der Fehlerwert in Form desjenigen Elementes, das hinzugefügt (XOR) werden muß, um das empfangene Element wiederherzustellen. Bei den Algorithmen, mit denen diese Parameter gefunden werden, müssen Gleichungssysteme gelöst werden.

Je mehr Fehler in dem empfangenen Datensatz auftreten, de-

sto größer muß das zu lösende Gleichungssystem sein. Wenn nur ein einziger Fehler auftrat, ist die Korrigieraufgabe trivial. Bei zwei Fehlern ist sie auch noch erträglich. Sie wächst aber bei mehreren Fehlern exponentiell an. In vielen Situationen sind außerdem Einelementfehler wesentlich häufiger als Mehrelementfehler. Aus diesem Grund werden Einfehler-, Zweifehler- und manchmal sogar Drei- und Vierfehler-Korrektoren direkt programmiert und als erstes auf die Aufgabe angesetzt. Erst wenn diese versagen, wird ein Allzweck-Korrektor, der wesentlich langsamer ist, dafür aber bis zum Korrekturvermögen hin jede Zahl von Fehlern korrigieren kann, eingesetzt. Das spart Korrekturzeit.

Der Allzweckkorrigierer arbeitet nicht sehr effizient. Er kann aber noch optimiert werden. Statt ein großes Gleichungssystem lösen zu wollen, um die Lage der einzelnen Fehler herauszubekommen, kann man z.B. eine *Chien*-Suche verwenden. Diese besteht einfach aus einer Schleife, die die Stelle eines jeden einzelnen Elementes Schritt für Schritt daraufhin untersucht, ob dort ein Fehler aufgetreten ist. Zur Ermittlung der Fehlerwerte wird der *Belekamp-Massey*-Algorithmus verwendet, der in der unten zitierten Literatur erklärt wird.

Angenommen, Sie wollen einen byte-orientierten Block der Länge 512 Byte gegen Fehler schützen. Wenn Sie Codeelemente von Bytegröße einsetzen, können Sie keine 512 Bytes überspannen. Sie müssen also den Block in kleinere Blöcke aufspalten. In vielen Systemen neigen die Fehler dazu, gleich im ganzen Schwall aufzutreten, so daß Mehrelementfehler benachbart sind oder zumindest dicht beieinander liegen. In solchen Fällen werden häufig kleinere Datenblöcke, jeder mit einem eigenen Fehlerschutz, miteinander verwoben. Die Elemente werden nach der Einfädungsmethode verschickt, je zu je von jedem Datensatz eines. Falls ein Fehlerschwall explosionsartig auftritt, teilen sich auf diese Weise die verschiedenen Datensätze die Last der Korrektur. Wir könnten beispielsweise den 512 Byte langen Datensatz in drei kleinere Datensätze von je um die 171 Bytes herum aufteilen und letztere miteinander verzahnen. Damit ist das Problem des zu großen Blocks gelöst und es hilft außerdem, die Korrektur-Effizienz zu steigern. Wenn ein 6-Byte-Fehlerschwall auftritt, wird dieser sich gleichmäßig auf die 3 Datensätze verteilen. Es ist aber viel effizienter, 2 Bytes in 3 Datensätzen zu korrigieren als 6 Bytes in einem.

Das Programmpaket zur Implementation von Reed-Solomon in Forth kann wegen seiner Größe nicht im vorliegenden Artikel aufgenommen werden. Es steht zum Downloaden bereit unter: <ftp://ftp.forth.org/pub/Forth/FD/1999/ReedSol.zip> Die .zip-Datei enthält fünf Einzeldateien:

- rsencode.txt* Erzeugung des endlichen Körpers und Codierer. Verifiziert auch den endlichen Körper.
- rsdecode.txt* Decodierer und einfache und doppelte Fehlerkorrektur. Enthält auch einige Disketten-Hilfsprogramme zum Lesen und Schreiben von Datensätzen.



Reed-Solomon-Fehlerkorrektur

rs correc.txt Der vollständige Fehler-Korrektor.
rs verify.txt Ein einfacher Code-Verifizierer zur
 Sicherstellung, daß alles funktioniert.
rs load.txt Lädt die obenstehenden Dateien.

Noch ein paar Anmerkungen

ENCODE holt ein Element vom Stack und codiert es. Die Ergebnisse finden sich im Datenfeld (im Array) REGISTERS wieder.

DECODE holt ein Element vom Stack und decodiert es. Die Ergebnisse finden sich im Array SYNDROMES wieder.

RESET-ENCODER und RESET-DECODER müssen aufgerufen werden, bevor mit einem neuen Datensatz begonnen wird.

DATABUF ist ein Array, das für die Daten, an denen man gerade arbeitet, verwendet werden kann.

ERROR? liefert den Flagwert TRUE, wenn das Array SYNDROMES mitteilt, daß ein Fehler aufgetreten ist. Es muß erst der gesamte Datensatz, einschließlich der FKC-Prüfelemente, decodiert werden, bevor SYNDROMES gültige Daten enthält.

CORRECT implementiert den Korrektur-Algorithmus. Es prüft, ob ein Fehler aufgetreten ist, und versucht dann eine Ein- und gegebenenfalls daraufhin eine Zweifehler-Korrektur. Wenn auch das nichts bringt, ruft es den vollen Fehlerkorrektur-Algorithmus auf. Es liefert den Flagwert TRUE, wenn die Fehlerkorrektur erfolgreich war. Die Daten in DATABUF werden vom Algorithmus aus CORRECT durch eine korrigierte Version ersetzt. Zu beachten ist, daß CORRECT eventuell den Flagwert TRUE liefert, obwohl die Daten gar nicht richtig korrigiert wurden, wenn der FKC-Mechanismus überlastet war. Wie oben gesagt, kann man das durch Einbau eines CRC-Schutzes kompensieren, oder indem man die erlaubte Zahl von Fehlern für den vollen Korrektor beschränkt. Wenn der FKC zum Beispiel acht Elementefehler korrigieren kann (16 Prüfelemente), und man beschränkt die Zahl der erlaubten Fehler auf 4 oder weniger und erklärt alle Datensätze mit fünf oder mehr Fehlern als fehlerhaft, dann wird die Wahrscheinlichkeit einer falschen Korrektur sehr gering ausfallen.

In der vorliegenden Implementation werden alle Elemente als CELLS gespeichert und die Prüfelemente werden stets von den Datenelementen getrennt im Array REGISTERS aufbewahrt. In der Praxis hängt man die Prüfelemente an die Daten an und versendet beides gemeinsam. In der vorliegenden Implementation gibt es keine Möglichkeit, mehrfache Datensätze zu verzahnen.

Bleibt zum Schluß noch zu sagen, daß mir die Interaktivität von Forth eine wertvolle Hilfe beim Verstehen dieses schwierigen Gebietes war. Die Möglichkeit, Arithmetik auf endlichen Körpern von der Tastatur aus zu betreiben, so als ob es ein spezieller Rechner wäre, und mit den Algorithmen einfach so herumzuspielen, war schon eine recht schöne Sache.

Literatur

- [1] *Theory and Practice of Error Control Codes*. Richard Blayhut. ISBN 0-201-10102-5 (1983).
- [2] *Practical Error Correction Design for Engineers*. 2.Aufl. Neal Glover und Trent Dudley. Zu beziehen von Cirrus Logic (303-466-5228).
- [3] "Error Recovery Codes", *Dr. Dobbs Journal* (Dez 1994). Bart de Cann. Gute Übersicht über FKC.

Dies & Das

Interessante URLs werden gewünscht ? Bitte schön, hier ist ein SEHR interessanter Link zu einem der bekanntesten Forther: Musik, Puppentheater und natürlich Brodies' Bücher, ein wenig über seine Familie und Vieles über einen 'alten Forther'...

Die Adresse von Leo ist:

<http://home.earthlink.net/~lbrodie/>

Was uns schon immer bewegte:

Eine der wichtigsten Fragen der Menschheit ist: "**Wie fange ich einen Elefanten?**".

Jede Berufsgruppe hat da ihre eigene Methode entwickelt:

z.B.: Mathematiker:

Mathematiker jagen Elefanten, indem sie nach Afrika gehen, alles entfernen, was nicht Elefant ist und ein Element der Restmenge fangen. Erfahrene Mathematiker werden zunächst versuchen, die Existenz mindestens eines eindeutigen Elefanten zu beweisen, bevor sie mit der Methode der gewöhnlichen Mathematiker als untergeordneter Übungsaufgabe fortfahren

Mathematikprofessoren beweisen die Existenz eines eindeutigen Elefanten und überlassen das Aufspüren und Einfangen eines tatsächlichen Elefanten ihren Studenten.

z.B.: Informatiker:

Informatiker jagen Elefanten nach unterschiedlichen Methoden, da es ja verschiedene Sprachen gibt, mit denen ein Such- und Fangmuster erstellt werden kann:

Für einen Algorithmus A folgendermassen:

1. Gehe nach Afrika
2. Beginne am Kap der guten Hoffnung
3. Durchkreuze Afrika von Süd nach Nord bidirektional nach Ost und West
4. Für jedes Durchkreuzen tue:
 - a) Fange jedes Tier, das Du siehst
 - b) Vergleiche jedes gefangene Tier mit einem als Elefant bekannten Tier
 - c) Halte an bei Übereinstimmung

Erfahrene Programmierer verändern den Algorithmus A, indem sie ein als Elefant bekanntes Tier in Kairo plazieren, damit das Programm auf jeden Fall korrekt beendet wird (terminiert).

Assembler-Programmierer bevorzugen die Ausführung des Algorithmus A auf Händen und Knien.

SQL-Programmierer verwenden folgenden Ausdruck:
SELECT Elefant FROM Afrika.

Natural-Programmierer lassen sich von ADABAS einen Elefanten bringen.

LOGO-Programmierer reiten durch Afrika auf ihrer Schildkröte.

Cobol-Programmierer tun dies auf einem Dinosaurier.

Basic-Programmierer bevorzugen jedoch einen mit Samt ausgepolsterten Einspanner, bei dem die Bremsen ständig angezogen sind.

C-Programmierer bestimmen zuerst mit sizeof() die nötige Speichermenge für einen Elefanten, versuchen diese zu allozieren, vergessen dabei das Ergebnis abzurufen und schießen dann mit wilden Pointern auf den Elefanten.

C++ Programmierer bestehen darauf, daß der Elefant eine Klasse sei und somit schliesslich seine Fangmethoden selbst mitzubringen habe. Und wenn der Elefant Afrika verlassen sollte, dann wird ja automatisch sein Destruktor ausgelöst.

Pascal-Programmierer markieren zuerst einen Punkt auf der Landkarte, schreiben dann END davor und träumen davon, daß Nikolaus Wirth von einem Elefanten totgetrampelt wird.

Modula-Programmierer importieren einen Elefanten aus/von einem Zoo.

LISP-Programmierer bauen einen Irrgarten aus Klammern und hoffen, das sich der Elefant darin verirrt.

Java-Programmierer erstellen je ein Applet für den Rüssel, die Ohren und den Schwanz und nennen das Ergebnis einen Javafanten.

HTML-Programmierer erstellen eine Frame- und eine Non-Frame-Version von Afrika und legen eine URL für einen Elefanten an, wenn einer irgendwo mal einen gefangen haben könnte.

Viren-Programmierer jagen Elefanten, indem sie eine Maus ans Kap der guten Hoffnung schicken und in Kairo auf die in Panik geratene Herde warten.

UNIX-Administratoren jagen Elefanten, indem sie den Busch

katalogisieren und dann über nfs nach /dev/afrika einbinden.

MVS-Operatoren jagen Elefanten nur, wenn ihnen jemand einen Job und ein Formular dafür zur Verfügung stellt und das Fach groß genug für den Output-Elefanten ist.

Windows-NT-Programmierer schiessen mit völlig ungeeigneten Gewehren in die völlig falsche Richtung und erklären dann, daß es ein Fehler am Elefanten sein muß.

Windows-95-Programmierer tun dasselbe, nur mit Pfeil und Bogen.

Zudem:

Microsoft kauft einen Elefanten aus dem Zoo in Seattle, kopiert ihn massenweise, redet aller Welt ein, daß jeder einen bräuchte, daß dieser die ideale Ergänzung zu MS Office sei und exportiert 14 Millionen Stück nach Afrika.

SAP-Systemingenieure erklären das erstbeste Tier zu einem Elefanten und passen ihre Vorstellungen eines Elefanten an dieses Tier an.

Nun, wie wär's mit:

Forth-Programmierer beschäftigen sich ausschließlich damit, geeignete Fangwerkzeuge zu erfinden.

Maschinen-Forth-Programmierer fangen keine Elefanten, sondern warten auf die Geburt eines Elefantenbaby, um es großzuziehen.

Viele Gruesse aus Freiburg,

Soeren Tiedemann

Forth verändert die Welt

Marx war der Meinung, die unterdrückten proletarischen Massen würden es nie schaffen, an die Produktionsmittel zu gelangen, es sei denn mit revolutionärer Gewalt. Engels unterstützte ihn, Lenin setzte seine Ideen in die Tat um. Das hat die Welt total verändert.

In Forth baue ich mir meine Produktionsmittel selbst. Das verändert die Welt ein zweites Mal.

Man nehme diese Beispiel dialektischen Denkens nicht allzu ernst.

beh

Texte, die Sie „irgendwo aufschnappen“, interessante Links zu forthingen Themen oder einfach Gedanken und Ideen, die Sie mit Anderen teilen oder diskutieren möchten: schreiben Sie diese Dinge der Redaktion der VD. Dann kann aus dieser Rubrik eine feste Einrichtung werden. Oder – noch besser – schreiben Sie uns, wenn Sie gerne eine eigene Rubrik in der VD selbst betreuen möchten !

fep

Einladung zur
Mitgliederversammlung
der Deutschen Forthgesellschaft e.V.

Am 16. April 2000, 9:00 Uhr

Haus Rissen
Internationales Institut für Politik und Wirtschaft
Rissener Landstr. 193
22559 Hamburg
Fon: 040 81907-0
Fax: 040 8190759
Mail: hausrissen@t-online.de

Tagesordnung:

1. Begrüßung der anwesenden Mitglieder
2. Wahl des Schriftführers
3. Wahl des Versammlungsleiters
4. Ergänzungen zur Tagesordnung
5. Bericht des Direktoriums
 - Zum Forthbüro (E. Woitzel)
 - Mitgliederentwicklung und Kassenstand (U. Woitzel)
 - Rund um die VD (F. Prinz)
 - Vorhaben des Direktoriums (Th. Beierlein)
6. Entlastung des Direktoriums
7. Wahl des Direktoriums
8. Verschiedenes

Entsprechend unserer Satzung können weitere Tagungsordnungspunkte auf Antrag einzelner Mitglieder von der Mitgliederversammlung durch Abstimmung auf die Tagesordnung gesetzt werden.

Für das Direktorium

Thomas Beierlein

Forth-Gruppen regional

Moers **Friederich Prinz**
Tel.: 02841-58398 (p) (Q)
(Bitte den Anrufbeantworter nutzen !)
(Besucher: Bitte anmelden !)
Treffen: (fast) jeden Samstag,
14:00 Uhr, **MALZ, Donaustraße 1**
47443 Moers

Mannheim **Thomas Prinz**
Tel.: 06271-2830 (p)
Ewald Rieger
Tel.: 06239-920 185 (p)
Treffen: jeden 1. Mittwoch im Monat
Vereinslokal Segelverein Mannheim e.V.
Flugplatz Mannheim-Neuostheim

München **Jens Wilke**
Tel.: 089-89 76 890
Treffen: jeden 4. Mittwoch im
Monat, **China Restaurant XIANG**
Morungerstraße 8
München-Pasing

mP-Controller Verleih

Thomas Prinz
Tel.: 06271-2830 (p)
micro@forth-ev.de

Gruppengründungen, Kontakte

Fachbezogen 8051 ... (Forth statt Basic, e-Forth)
Thomas Prinz
Tel.: 06271-2830 (p)

Forth-Hilfe für Ratsuchende

Forth allgemein
Jörg Plewe
Tel.: 0208-49 70 68 (p)

Jörg Staben
Tel.: 02103-24 06 09 (p)

Karl Schroer
Tel.: 02845-2 89 51 (p)

Spezielle Fachgebiete

Arbeitsgruppe MARC4 Rafael Deliano
Tel./Fax: 089-841 83 17 (p)

FORTHchips Klaus Schleisiek-Kern
(FRP 1600, RTX, Novix) Tel.: 040-375 008 03 (g)

F-PC & TCOM, Asyst Arndt Klingelberg, Consultants
(Meßtechnik), embedded akg@forth-ev.de
Controller (H8/5xx// Tel.: ++32 +87 -63 09 89 pgq
TDS2020, TDS9092), (Fax -63 09 88)
Fuzzy

KI, Object Oriented Forth, Ulrich Hoffmann
Sicherheitskritische Tel.: 04351 -712 217 (p)
Systeme Fax: -712 216

Forth-Vertrieb volksFORTH / ultraFORTH
RTX / FG / Super8 / KK-FORTH
Ingenieurbüro Klaus Kohl
Tel.: 08233-3 05 24 (p)
Fax : 08233-99 71
mailorder@forth-ev.de

Forth-Mailbox (KBBS) 0431-533 98 98 (8 N 1)
Sysop Holger Petersen
hp@kbbs.org
Tel.: 0431-533 98 96 (p) bis 22:00
Fax : 0431-533 98 97
Helsinkistraße 52
24109 Kiel



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 ?

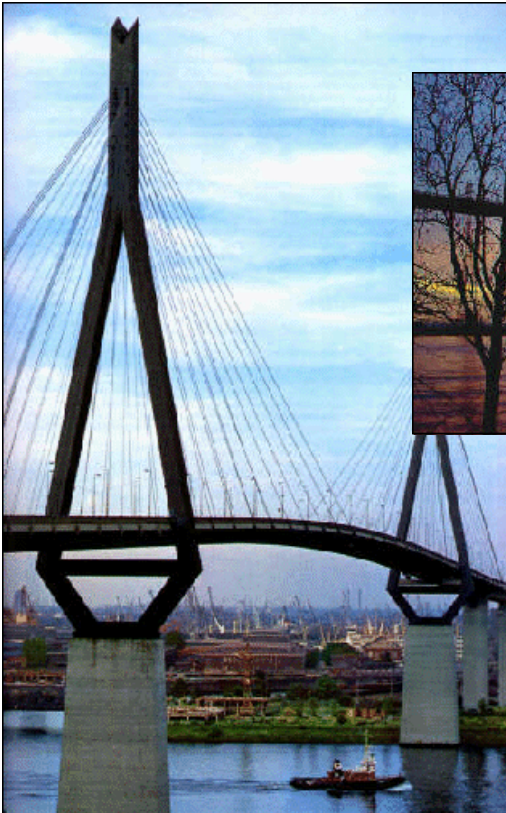
Schreiben Sie einfach der VD - oder rufen Sie an - oder schicken Sie uns eine E-Mail !



Hinweise zu den Angaben nach den Telefonnummern:

Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich

Die Adressen des Büros der Forthgesellschaft und der VD finden Sie im Impressum des Heftes.



Forth Jahrestagung 2000

- HAMBURG -

vom 14.-16.4.2000

mit der Option "13.4."

Es beginnt mit Kaffee am frühen Nachmittag des Donnerstag oder Freitag
und endet mit dem Mittagessen am Sonntag.

Ich habe für FR-SO 10 Doppel- und 10 Einzelzimmer bestellt,
für den DO 8 Doppel- und 5 Einzelzimmer.

Die Tagungskosten für die

Mitglieder / Nichtmitglieder

Teilnehmer	310 / 340
Ermäßigt	260 / 260
Gäste	260 / 260
Donnerstag	120 / 120
Einzelz.	25 / 25 pro Tag zusätzlich

Diese Preise gelten bei Anmeldung bis 20.2.2000. Danach ist ein
"Spätanmelderzuschlag" von einmalig DM 50,- zu zahlen.

Klaus Schleisiek

SEND Signal-Elektronik und Netz-Dienste GmbH
Stubbenhuk 10
D-20459 Hamburg
Tel: +49 40 375008-03
Fax: +49 40 375008-93

