Vierte Dimension
Volume III/Nr.2 Juni 1987

VIERTE DIMENSION

VAR
CASE
LAUFZEITEN
FORTHCHIPS
IEEE FORTH?
EUROFORML 87

FORTH MAGAZIN

5,00 DM

Fifth 2.0
von Cliff Click & Paul Snow

Fifth ist eine auf Forth basierende Programmier-Umgebung. Diese besteht im wesentlichen aus einem Interpreter, einem Compiler, einem Text Editor und einem Dictionary Editor.

Fifth besitzt folgende Eigenschaften:

- Dictionary Editor erlaubt Änderungen des Dictionaries (ändern, umordnen, einfügen, löschen,...)
- Automatisches Compilieren von geänderten Programmteilen
- Baumstrukturierte Modularisierung
- Fullscreen Editor mit 80 Zeichen pro Zeile
- Source Level Tracing und Breakpoints
- 32-bit Adressen und Arithmetik
- 8087 Floating Point Unterstützung
- MS-DOS File Interface
- Graphik Funtionen
- Online Help
- Turnkey Applikationen möglich
- Erzeugt Maschinencode
- Enthält F83 Public Domain Assembler

Verfügbar ist **Fifth** momentan für den IBM-PC und Kompatible und den TI-PC unter MS-DOS. Weitere Versionen, speziell für 68000er Systeme, befinden sich in Vorbereitung.

Fifth wird als Shareware Produkt auf 1 Diskette ausgeliefert. Diese enthält das komplette System, diverse Demos und die Dokumentation in Englisch. Sie erhalten die Diskette per Nachnahme, oder nach Überweisung von DM 25,- auf das Konto:

Rainer Mertins	Konto: 539553-205
Antilopenstieg 6a	PGA Hmb, BLZ 200 100 20
2000 Hamburg 54	

Ähnlich wie das volksFORTH83 und das **F83** wird auch **Fifth** nicht von der **Forth Gesellschaft** eV selbst vertrieben, sondern von einigen Mitgliedern, die die Arbeit des gemeinnützigen Vereins **Forth Gesellschaft eV** damit unterstützen wollen. Wenn Sie mehr über Forth wissen wollen, dann sollten auch Sie Mitglied werden!

(Stand: Juni 1987)

INHALT

3 Editorial
4 EuroFORML Konferenz 1987
6 Impressum und Anleitung für Autoren
7 Nachrichten
9 Zuschriften
11 Forth Literatur, Prinz
12 Deutschsprachige Forth Bücher
13 Lösung der Aufgabe GELDBETRAG
16 INPUT# Nachlese
17 Neues vom forthTREE, Pauck
19 Quelltextdokumentation, Pennemann
22 Laufzeitmessung von Forthworten, Pennemann
29 Strukturierte Datentypen, Hoffmann
31 Zur Geschichte der Forthchips
32 FORTHkit und Computer Cowboys
34 Multiplikation mit Zwei, Kalus
35 A Forth Dictionary, Berlev
38 State of the Standard, Ouverson
40 Ultimate CASE Satement, Wil Baden
43 Floating Point Arithmetic, Wilton
46 Die Aufgabe
47 Gruppen
48 Rückseite

INVITATION & CALL FOR PAPERS

euroFORML conference
on the FORTH programming language
and FORTH processors

from September 18th through 20th 1987
at Stettenfels Castle, Federal Republic of Germany

sponsored by
"Forth Gesellschaft eV, FRG"
and
"Forth Interest Group Inc, USA"

euroFORML will be an international meeting of computer practitioners using FORTH as a problem solving tool. Lectures, workshops and presentations are planned to demonstrate techniques and problem solving strategies that have proved uesful.

This years conference will specifically focus on FORTH in hardware and the possibilities that are opened up by the quantum leap in execution speed of the new Forth processors.

The conference will take place at Stettenfels Castle (12th century) on top of Untergruppenbach in the vincinity of Heilbronn near Stuttgart, FRG. The castle can accomodate 60 guests and has room for 110 conference participants. Reservation for bed and breakfast/hotels nearby can be arranged.

The conference language will be English and FORTH, of course. The conference is supposed to be self organizing , ie. there will not be a strict agenda prior to the beginning of the conference. If you want to present your ideas you may choose one of the following formats:

Paper presentation
You will present a paper in a 10 minute talk in front of the whole group with the possibility to get immediate feedback.

Poster presentation
You will be assigned a "poster space" where you can present your ideas to a small group of people in a seperate room. This is especially useful for demonstrations of hard- and software.

Workshops
You may organize or participate in workshops which will be organized at the beginning of the conference, depending on demand.

An English language proceedings will de published after the conference; papers to be included in the proceedings will be handed out to the paticipants in photo copied form at the beginning of the conference.

Registrations should be mailed by August 1st. A DM 200,- deposit per person is required. (Money order in German funds, eurocheques in foreign funds or transfer to Postgiro account# 5632 11 - 208, bank code 200 100 20). The full amount is due at the beginning of the conference. Space is limited and you are assigned on a first-come, first served basis.

This year we reserve 1/3 of the available space for students at a reduced rate. Guests may take their meals at the castle but they can only be accomodated if space permits. You may choose to bring your own tent and camp on the camping grounds in front of the castle.

Participant DM 640,- (600,-)
(accomodation in the castle, three meals per day and conference proceedings)

Guest DM 490,-
(accomodation in the castle and three meals per day)

Participant DM 490,-
(lodging outside the castle, three meals a day and conference proceedings)

Student (limited openings) DM 320,- (300,-)
(accomodation in the castle, three meals per day and conference proceedings)

(Reduced rates for members of Forth Gesellschaft)

Author instructions

Papers to be presented at the conference (to be included in the proceedings) will have to be mailed to the Forth Gesellschaft eV no later then September 1st 1987 in camera ready form. The format is DIN A 4 (letter size) with a margin of 2,5 cm (1.5") on all sides. Every page should containe a page number and the authors name. Papers should not exceed 15 pages. Code should be accompaniedby shadow screens.

For reservations and conference papers write or call:

C.D. Osten
Gneisenaustr. 23 / D-2000 Hamburg 20 / FRGermany
(49) (40) 422 1694 or (49) (40) 490 5195

Direct your questions which are not related to the conference to:

Forth Gesellschaft eV
Friedensallee 92 / D-2000 Hamburg 50 / FRGeermany
(49) (40) 390 4204 Tuesday 18:00-20:00

Anleitung für Autoren
Das FORTH MAGAZIN 'Vierte Dimension' veröffentlicht originale Arbeiten, Berichte und Bibliographien, die in Bezug zur Programmiersprache FORTH stehen. Manuskripte können an das Büro der Forth Gesellschaft geschickt werden, oder direkt an die REDAKTION des Forth Magazins, zZ.:
Michael Kalus, Präsidentenstr.40, D-5830 Schwelm

Die Arbeiten sollten folgendes enthalten:

1. TITELSEITE
   Eine eigene (erste) Seite sollte enthalten: Titel, Autor und Institut oder Bertieb, in dem oder für den die Arbeit angefertigt wurde.
2. ZUSAMMENFASSUNG
   Die Zusammenfassung von 50-100 Worten sollte Absicht, Methoden, Ergebnisse und Schlußfolgerungen der Arbeit enthalten und für sich genommen bereits verständlich sein.
3. SCHLÜSSELWORTE
   Etwa fünf Schlüsselworte sollten ausgesucht werden, für die die Arbeit relevant ist.
4. TEXT
   Wenn möglich sollte der Text in klassischer Form aufgebaut sein, dh in einer kurzen Einleitung über das Ziel der Arbeit informieren, Materialien und Methoden hinreichend genau wiedergeben, über die Entwicklung der Ergebnisse oder Systeme berichten, diese diskutieren und Schlüsse ziehen.
5. DANK
   Für Hilfen oder Rat, technische Mitarbeit, Materialien usw. sollte Dank in einem eigenen Abschnitt am Ende der Arbeit ausgesprochen werden.
6. QUELLENANGABEN
   Die Quellenangaben sollen auf einer eigenen Seite getippt sein, um sie für das Layout gesondert verkleinern zu können, sollen alphabetisch nach Autoren geordnet und durchnummeriert sein. Im Text beziehen sich die Nummern in runden Klammern auf diese Liste. Zeitungsartikel sollen mit den Namen und Initialen von allen Autoren, dem vollen Titel des Artikels, dem Namen der Publikation, der Rubrik, dem Erscheinungsjahr und der Nummer der erste und letzten Seite des Artikels genannt werden. Bücher sollen mit dem Namen des Autors, dem vollen Titel, Ausgabe, Erscheinungsort, Herausgeber und Jahr genannt werden.
7. ILLUSTRATIONEN
   Die Illustrationen sollen auf das unbedingt Notwendige beschränkt werden und keine Daten enthalten, die besser in Tabellenform wiedergegeben werden können. Diagramme und Kurvenverläufe sollten als Schwarz-Weiß-Zeichnungen kopiergeeignet sein. Bilder sollten ebenfalls Schwarz-Weiß, scharf und kontrastreich sein. Anleitungen für das Layout oder Anmerkungen zum Text sollten auf einem aufgelegten Transparentblatt und nicht auf der Vorlage selbst gemacht werden. Alle Illustrationen sollten auf ihrer Rückseite dem Text entsprechend nummeriert sein.
8. TABELLEN
   Tabellen sollen auf einer eigenen Seite getippt sein, eine Nummer und eine Überschrift tragen und im Text angesprochen werden. Jede Spalte sollte einen Namen tragen. Senkrecht gestellte Beschriftung möglichst vermeiden.
9. QUELLCODE
   Quellcode soll auf einer eigenen Seite getippt sein, eine Seiten- oder Screen-Nummer und eine Überschrift tragen und im Text angesprochen werden. Die Kommentare zum Code sollen zeigen, WAS der kommentierte Abschnitt ausführt, also den Sinn wiedergeben. Bei besonderen Programmiertechniken kann der Kommentar auch das WIE näher beschreiben oder begründen. Bitte stets mit einem kräftigen Farbband drucken. Bitte nur Listings von getesteten Programmen einsenden. Programmfehler fallen auf den Autor zurück.

Verwenden Sie eine normalgroße, nicht zu dünne Schrift. Verwenden Sie möglichst ein frisches Farbband. Verwenden Sie nicht mehr als 80 Spalten und 72 Zeilen auf einer DIN A4 Seite. Die Beiträge werden überarbeitet, um die Kommunikation zwischen Leser und Autor effektiver zu machen und um Mehrdeutigkeiten zu vermeiden. Wenn ausgedehnteres Edieren nötig ist, erhält der Autor vor der Wiedergabe den Beitrag zur Korrektur und Zustimmung zu Verbesserungsvorschlägen zurück. Die Layouts werden nicht mehr zur Prüfung durch die Autoren vorgelegt. Autoren erhalten 5 kostenlose Exemplare des FORTH MAGAZIN. Auf Wunsch auch mehr, falls der Vorrat reicht. Manuskripte können auch per DFü übergeben werden.

IMPRESSUM

Titel: FORTH MAGAZIN 'Vierte Dimension'.
Zeitschrift der Mitglieder der Forth Gesellschaft eV.
Herausgeber: Forth Gesellschaft eV.
Forth: Klaus Schleisiek und die Mitglieder des Review-Boards sowie alle namentlich genannten Autoren.

Redaktion: Michael Kalus, Tel: 02336-82204
Präsidentenstraße 40, 5830 Schwelm
Erscheinungsweise: Ein Heft je Quartal.
Redaktionsschluß: Der mittlere Quartalsmonat.
Auflage: 500 Stück europaweit.

Druck:

Das Forth Büro

Forth Gesellschaft eV, Friedensallee 92, 2000 Hamburg 50

Tel.: 040-3904204

Jeden Dienstag von 18:00 bis 20:00 Sprechstunde.
Zu allen anderen Zeiten ist der TREE angeschlossen.

Postgiroamt Hamburg, Kto: 563211-208, BLZ 20010020

NACHRICHTEN

*** Spaltung bei NOVIX

Drei Gründungsmitglieder der Firma NOVIX, John Golden, John Rybel und Bob Murphy, haben Novix verlassen. Sie entwickeln in ihrer eigenen Firma QSD-Inc. den Forthchip weiter, angeblich im Auftrag eines einzigen Kunden...

*** F83 für NC4000

Mike Perry hat die Implementation seines F83-Forthsystem für den NC4000 fertiggestellt. Es soll weiterhin in der public domaine bleiben und NOVIX will es mit dem Forthchip vertreiben. Das Forth des Chips selbst (CM-Forth) ist ein MinimalForth, Opcodes gleichzusetzen, und ohne jeden 'höheren Komfort'. Die Implementation des F83 auf NC4000 verbindet die umfangreiche Programmierumgebung des F83-Systems mit der ungeheuren Schnelligkeit des Forthchips.

*** ANSI oder IEEE Forth Standard?

Die Versuche, Forth in einem neuerlichen 'Standard' festzuhalten, dauern an. Dieses Mal sind es die 'großen' Namen im Geschäft, die auf ein ANSI Forth hinarbeiten. Insbesondere die FORTHInc. USA, wieder vereint mit Chuck Moore, treiben dies voran und vorbei am alten 'Forth Standard Team'. Mit dabei diesmal auch IBM. Lesen sie dazu auch den Beitrag von Martin Ouverson (aus: Forth Dimensions, VIII/6, April'87).

*** Preisnachlaß für Layouts.

Alle Mitglieder der Forthgesellschaft, die privat eigene Platinen herstellen wollen, können jetzt ihre Schaltungen zu Sonderpreisen entflechten und die Layouts plotten lassen. Wie es dazu kam, lesen Sie in der Zuschrift von Uwe Tams.

*** Neuer Forth-Chip von Charles Moore?

Unbestaetigten Verlautbarungen einer zuverlaessigen Quelle zufolge arbeitet Chuck Moore z.Z. an einem neuen FORTHchip. Und zwar mit einem, im Vergleich zum NC4000 veraenderten, 32 bit Design. (aus: TREE 1/87)

*** NX4-Board

In der Februar Ausgabe der Zeitschrift ELECTRONICS & WIRELESS WORLD begann eine Reihe von Artikeln ueber den Novix NC4000 und ein Low-Cost Experimentierboard. Diese NX-4 genannte Platine ist etwa mit dem FORTHkit von Chuck Moore oder dem Novix-EB1 von Forth-Systeme A.Flesch vergleichbar.

Die Wahl der Zugriffszeit fuer das Main-Memory ist jedoch ueber eine recht interessante, diskret aufgebaute Logik (und nicht wie beim EB-1 ueber ein PAL) vielseitig realisiert. Daher laesst sich das NX-4 wohl besonders einfach mit unterschiedlichster Peripherie verbinden. Die Schaltplaene des Boards und ein Beispiel fuer das Interfacing von A/D- und D/A-Wandlern sind abgedruckt. Diese sind auch fuer FORTHkit und EB-1 Besitzer interessant.

Das Board ist in einer limitierten Auflage von 300 Stueck zum Stueckpreis von 286 Pfund erhaeltlich bei Golden River Ltd., Churchill Road, Oxfordshire OX67XT.

*** Forth-Chip der John Hopkins Universität

Das Applied Physics Laboratory der John Hopkins University (JHU/APL) hat einen FORTHchip entwickelt. Er ist der erste FORTHchip, der eine 32-bit Architektur hat. Der eigentliche Anlass, diesen Chip zu entwickeln war zu zeigen, dass es moeglich ist, einen leistungsfaehigen 32-bit Prozessor in Custom VLSI Technologie mit verhaeltnismaessig wenig Aufwand zu entwerfen, zu implementieren und zu testen.

Der Prototyp ist mit ca. 18000 Transistor-Funktionen auf 73 mm2 in 4 micronSilicon-On-Sapphire Technologie realisiert. SOS hat gegenueber NMOS oder CMOS bessere Eigenschaften fuer den Einsatz im Weltraum. Die Spezifikation der Architektur benoetigte ca. 5 Mann-Monate. Logik-Entwurf, Layout und Simulation benoetigten weitere 9 Mann-Monate.

Im Gegensatz zum Novix-Chip hat der JHU/APL-Chip eine weniger ausgefallene Architektur. Parameter- und Return-Stack werden in das Main-Memory gemapped. Aber auch er ist ein Vertreter der RISC-Philosophie und fuehrt die meisten Forth Worte in 1-2 Taktzyklen aus.

In den Proceedings der FORML'86 sind mehrere Paper veroeffentlicht, die den Chip ausfuehrlich beschreiben.

*** Novix Club

Wer Interesse speziell am NC4000 hat, kann sich dem Novix Club Deutschland, Michael Hermann, Roderstr 10, 7730 VS-Villingen anschließen.

Sie geben deutschsprachige Info's heraus, halten Kontakt zum NOVIX Register und dem amerikanischen Club. Dort gibts auch die neuesten Info's ueber den NC6000, der im Januar '87 eigentlich schon in Mustern verfuegbar sein sollte.

*** Forth Systeme Flesch

Die Firma ist umgezogen nach Breisach. Die Firma vertreibt NOVIX in der BRD. Dazu hat sie eine eigene Experimentier-Platine für den NC4000 im Angebot, die zZ. etwa 600,-DM kostet. Es ist das NOVIX-EB1. Es ist ein fertig aufgebauter Europakarten Rechner (160x100) mit VG-Leiste, 8K-Worte RAM, 8K-Worte Eprom, serielle Schnittstelle, PAL-Speicherdekodierung, sowie der Moeglichkeit auch 32K8 RAMS einzusetzen.
Über Flesch können auch alle anderen NOVIX-Produkte aus USA bezogen werden - PC4000, Beta Board auch mit Harddisk, V4000 CPU Board für VME, Chips etc.
Forth Systeme Flesch, Postfach 1103, Kühnheimerstr.11, D-7814 Breisach, Tel: 07667-551.

*** FORTHkit

Das von Chuck Moore (Computer Cowboys) angebotene Entwicklungs-board fuer den Novix-Chip heisst FORTHkit. Die Beschreibung finden sie weiter hinten im Heft. (aus: TREE 5/87)

*** N e i n n e i n n e i n .

Nein, EURO FOR ML steht keinesfalls für "Europa den Marxisten-Leninisten" !!! Da irren sie sich. Das Signet findet sich auf den FORML PROCEEDINGS, made in USA. Für Europa angepasst von Thomas Prinz.

ZUSCHRIFTEN

Noch ein Wort zum Artikel von R. F. Illyes VDII,4,
"A Fast High-Level Floating Point".

Da heisst es auf Seite 17 oben

```
Create PL  3, here ....
```

und einige Zeilen weiter

```
: Tens  2* 2* Literal + 2@ ;
```

Offensichlich soll 'Here' als Literal in 'Tens' compiliert werden. Diese Vorgehensweise fuehrt bei einigen 4th-Systemen zu einer Fehlermeldung, weil die Stacktiefe vor und nach der Compilation von 'Tens' nicht den selben Wert hat. Aber auch aus stilistischen Erwaegungen sollte hier besser mit '[ ]' gearbeitet werden.

Also: `: Tens  2* 2*  [ PL WortBreite + ] literal + 2@ ;`

'Here' in 'Create Pl...' wird jetzt natuerlich wegelassen.

B. Molte, Lemgo, Tel. 05231-72823

*

Tams
An die Forthgesellschaft. Angefangen hat alles mit dem AIM65 von Rockwell und Forth. Forth verdanke ich so manches. Heute erstelle ich Platinenlayouts beruflich. Aus alter Liebe zum Forth bin ich Mitglied in der Forthgesellschaft und möchte den Verein durch einen praktischen Beitrag fördern.

An Vereinsmitglieder gebe ich Platinen-Layouts zu rein kostendeckenden Preisen ab, sofern sie zunächst zu privaten Zwecken verwendet werden. Wer Fragen dazu hat, kann mich gerne anrufen.

Ich entflechte und erstelle Layouts im Maßstab 1:0,5 bis 1:2,5 stufenlos und farbig, dazu den Bohrplan, den Bestückungsplan, die Bauteileliste und die Verbindungsliste ausgehend von Schaltungszeichnungen und Bauteilvorgaben. Das Layout wird auf Papier oder Transparentpapier geplottet geliefert. Abgerechnet wird nach Stundensatz: 15,-DM/Std. Wer will, kann das Layout auch auf PVC-Folie bekommen. Dafür muß ich dann allerdings 2,50DM (A5) bis 8,-DM (A3) extra berechnen.

Dieses Angebot soll Neuentwicklungen fördern helfen. Eine gewerbliche Nutzung ist zunächst einmal nicht gestattet. Doch natürlich wünsche ich jedem auch den vollen Erfolg seiner Produktion. Wenn jemand schließlich mit seiner Entwicklung ins Geschäft kommt, gebe ich die endgültigen Layouts für einen Zuschlag von 10% zur freien Verwendung ab. Natürlich ist das nur gegen Mitgliedsnummer zu machen. Ich wünsche allen ein gutes Gelingen ihrer Projekte.
Uwe Tams, Westring 273, 23 Kiel 1, 0431-180975

*

Soeder
Frage: Müssen wir es den Amerikanern und ihrer 'Forth Dimensions' nachmachen und die Bände der VIERTEN DIMENSION mit römischen Zahlen bezeichnen? Ich bin für arabische Numerierung - was macht ein Datenbanksystem mit römischen Ziffern? Andreas Soeder

Antwort des FORTH MAGAZIN: Es ist üblich, Zeitschriften durch Titel, Jahrgang (Volume), Heftnummer und Datum zu kennzeichnen. Römische Zahlen für den Jahrgang sind dabei reine Geschmacksache. Der menschliche (!) Leser kann sie von den laufenden arabischen Heftnummern klar unterscheiden. Das finde ich hübsch und möchte dabei bleiben. Aber das Datum, tja, das fehlte bisher in unserer Titelei!

Also, ab sofort wird der Druckmonat der Ausgaben mit angegeben; zB so: Vierte Dimension, Volume III, Heft 2, Juni 1987. Im Briefverkehr oder in Beiträgen hier in der VD ist auch die Kurzform üblich, zB: VD 2/87. Bei der Kurzform wird also der Jahrgang einfach weggelassen. Das geht, da die Jahrgänge und Kalenderjahre der VD zusammenfallen. (mk)

*

Forth Progamm Editior
Kein Leserbrieft im eigentlichen Sinn, aber eine oft gehörte Frage ist diese: Warum gibt es in Forth keinen guten Editor?

Die Antwort ist einfach: Es gibt sie! Aber erstens sind nicht alle auch gleich public domaine und zweitens sind die Geschmäcker verschieden. Und es gibt zudem jede Menge Textprogramme, die gut genug sind, um Forth-Programme einzutippen; und damit lassen sich die Files zudem noch verwalten etc. Wozu also sich die Mühe machen? Es ist einfacher, solche Textfiles über Filter vom Forth aus zu lesen.

Versuchen sie mal, einen Editor zu schreiben, nur so einen 'ganz kleinen full-screen-editor' für 1K-Blöcke, und sie werden sehr schnell einsehen, daß das keine triviale Sache ist, vor allem, wenn man wirklich versucht, unabhängig von der jeweiligen Hardware zu bleiben. Ein Zeileneditor hingegen, wie ihn die meisten Forthsysteme haben (zB F83), ist simpel genug, um seine Arbeitsweise sofort zu erkennen. Und so sind sie leicht an diverse Hardware anzupassen. Ich finde diese Zeileneditoren daher noch lange nicht 'überholt'.

Und doch gibt es in Forth geschriebene Texteditoren. Wer so etwas sucht, schaue sich mal das volksFORTH-83 und den Editor dazu von D.Weineck auf dem Atari an! Einen Screen-Editor bietet auch die CP/M Version des Volksforth. Hier kann man sehen, wie es gemacht werden kann. Und wer nun noch Wünsche offen hat, kann aus dieser public domaine Grundlage ja mal was noch Tolleres machen. Für eine Arbeitsgruppe darüber finden sich bestimmt eine Handvoll Leute.

PS: Im TREE fand ich gerade ein Programm mit dem Namen FULL-SCREEN-EDITOR. Kritiken dazu erbeten. (mk)

*

Redeker
Lieber Michael Kalus, zunächst einmal vielen Dank für deine Vermittlertätigkeit. Der Kontakt zu Ulrich Hoffmann (Volksforth unter CP/M; die Red.) hat mir viel gebracht. Nun möchte ich möglichst viele der speziellen Funktionen meines CPC464 auch von Forth aus benutzen. Wer ähnliches vorhat wie ich, schreibe mir bitte zum Programm- und Erfahrungsaustausch.
Markus Redeker, Julie-Postel-Str.5, 4352 Herten

*

Hoffmann
Endlich habe ich es geschafft, zumindest teilweise, die volksFORTH Files auf Apple CP/M zu kopieren. Ein kleines Problem hat sich ergeben, da der Sourcetext nämlich 128KB groß ist. Er paßt daher nicht auf eine Diskettenseite... Ulrich Hoffmann, Kiel, 04307-6869

(Das Volksforth ist in der alten AppleII Welt angekommen. Vielen Dank Ulrich, läuft prima. Auch die Quelle ist jetzt da. mk)

* * *

THE JOURNAL OF FORTH APPLICATION AND RESEARCH

Stand : 07.05.1987 "The Journal of FORTH Application and Research"

Jahr	Vol./Nr.	Org.	Kop.	Abg.	Thema
1983	1 / 1	2	15	2	
1983	1 / 2	2	15	2	Data Structures and Implementation
1984	2 / 1	2	15	2	FORTH Machines - Michael K. Starling
1984	2 / 2	2	11	2	Real Time Systems
1984	2 / 3	2	15	2	Enhancing FORTH
1984	2 / 4	2	11	2	FORTH on Large Machines VAX,68k,80xxx
1985	3 / 1	2	17	2	
1985	3 / 2	2	--	--	1985 Rochester FORTH Conference
1986	3 / 3	2	17	2	Application Languages
1986	4 / 1	2	--	2	Expert Systems in FORTH - Jack Park
1986	- / -	-	--	--	
1987	4 / 3	1	--	--	?????
1987	- / -	-	--	--	
1987	- / -	-	--	--	
1987	- / -	-	--	--	
1987	- / -	-	--	--	

Deutschsprachige Forth Bücher

Beilstein, H.-W.: Wie man in FORTH programmiert,
1986, ca. 300 S., kart., ca DM 33,-, 3-8023-0165-X (Vogel);

Birkenmeyer, R.: Programmiersprache der 4. Generation: FORTH,
1986, 160 S., kart., DM 28,50, 3-921608-38-4 (Elektor);

Brodie, L.: Programmieren in FORTH,
1984, 327 S., kart., DM 48,-, 3-446-14070-0 (Hanser);

Brodie, L.: In Forth denken,
1986, 285 S., kart., DM 48,-, 3-446-14334-3 (Hanser);

Chirlian, P.M.: Der Einstieg in FORTH,
1985, 337 S., kart., DM 58,-, 3-89090-085-2 (Markt & Technik);

Flögel, E.: FORTH Anwendungen,
1984, 205 S., kart., DM 49,-, 3-88-963-200-9 (Hofacker);

Flögel, E.: FORTH Handbuch,
1982, 184 S., kart., DM 49,-, 3-921682-6 (Hofacker);

Glasmacher, P./Kiesenberg, D.: FORTH Handbuch,
1984, 180 S., kart., DM 46,80, 3-923608-12-8 (Kiesenberg);

Glasmacher, P./Kiesenberg, D.: Programmieren in FORTH,
2 A., 1984, 172 S., kart., DM 46,80, 3-923608-00-4 (Kiesenberg);

Goppold, A./Bouteiller, R.: Forth: Ein Programmiersystem ohne Grenzen,
1985, 220 S., geb., DM 29,80, 3-924690-02-2 (Ed. Aragon);

Hogan, Th.: Forth - ganz einfach,
1985, 77 S., kart., DM 29,80, 3-528-04292-3 (Vieweg);

Knecht, K.: Einführung in FORTH,
1984, 218 S., kart., DM 58,-, 3-922120-73-3 (Markt & Technik);

Monadjemi, P.: FORTH für Fortgeschrittene,
1985, ca. 200 S., kart., DM 49,-, 3-89011-111-4 (Data Becker);

Monadjemi, P.: Forth Tips & Tricks,
1985, kart., DM 49,-, 3-89011-111-4 (Data Becker)

Monadjemi, P.: Das Trainingsbuch zu FORTH,
1984, 300 S., kart., DM 39,-, 3-89011-055-X (Data Becker);

Reymann, J.: FORTH,
1985, 94 S., kart., DM 9,80, 3-442-13128-6 (Goldmann);

Winfield, A.: Alles über FORTH;
1985, 140 S., kart., ca. DM 32,-, 3-88793-137-8 (Idea);

Winzer, Th.: FORTH,
1986, 140 S., kart., DM 39,50, 3-907007-04-2 (Informa);

Zech, R.: Die Programmiersprache FORTH,
2. A. 1985, 333 S., geb., DM 68,-, 3-7723-7262-7 (Franzis);

Zech, R.: Forth 83,
1987, ca. 320 S., geb., DM 78,-, 3-7723-8621-0 (Franzis);

GELDBETRAG

Lösung der Aufgabe aus VD 1/87

Geldbetrag. Das war die Aufgabe aus dem letzten Heft. Die hier vorgestellte Lösung von B. Molte haben wir aus mehreren richtigen ausgewählt, weil hieran sehr schön zu sehen ist, was 'faktorisieren' heißt. Das Problem der Zahlenausgabe wurde deutlich gemacht und geschickt in Komponenten von allgemeinerer Gültigkeit zerlegt. GELDBETRAG - auf diese Weise elegant gelöst.

R.Mensing gibt in seiner Lösung die WAEHRUNG als aufgeschobene Definition eines String nach der ZAHL aus. So ist GELDBETRAG nicht an DM allein gebunden. (ZAHL definiert er im Prinzip wie B.Molte auch, der Code soll daher nicht zweimal abgedruckt werden. Das Beispiel von R.Mensing wurde daher gekürzt.)

Allen anderen Einsendern ebenfalls herzlichen Dank, auch sie haben die Aufgabe bestens gelöst und sich wacker mit den Primitives <# # HOLD #> geschlagen.

```
( Währungen zuweisen)  DEFER WAEHRUNG
: MARK     ( -- ) ." DM" ;      : GULDEN  ( -- ) ." hfl" ;
: DKRONEN  ( -- ) ." dkr" ;     : DOLLAR  ( -- ) ." $" ;

' MARK is WAEHRUNG     ( ... usw. Und schließlich:)

: GELDBETRAG  ( d n x y --)  AT ZAHL WAEHRUNG CR ;
```

R.Mensing

Ich habe vor einiger Zeit eine Zahleneingabe im Taschenrechner-format in Forth geschrieben. Die dabei verwendeten Forth-Worte sind hier zum Teil aufgefuehrt. Die Aufgabe aus der VD III,1 laesst sich mit diesem Wortstamm leicht realisieren, bzw. geht noch um einiges darueber hinaus (Tausenderpunkte, variable Dezimalstellen).

B. Molte, 4920 Lemgo, Steinstoss 24, Tel. 05231/72823

```
: type/r  ( addr n1 n2 -- )  \ Gibt den String (addr n1) rechts-
 over - spaces  type ;       \ buendig im n2-breitem Feld aus.


: .DM  ( d -- addr n )
 saveSign
 <# ascii M hold  ascii D hold
    2 #Frac  #Int  addSign  #> ;


: Geldbetrag  ( d n2 x y -- )    \ Gibt d an x-y im n2-breitem
 at  >r  .DM  r> type/r ;        \ Feld rechtsbuendig aus. n2
                                 \ wird ignoriert, wenn n1>n2 ist
```

B. Molte, 4920 Lemgo, Steinstoss 24, Tel. 05231/72823

Glossar der verwendeten Worte:

MU/ ud un -- ud-q
Dividiert ud durch un und gibt ud-q als Quotienten zurueck.

DStellen ud -- n
Berechnet aus ud die Anzahl der Dezimalstellen.

TPunkte ud -- n
Berechnet aus ud die Anzahl der moeglichen Tausender-Punkte.

#Int ud1 -- ud2
Erzeugt aus ud1 einen Ausgabestring mit Tausender Punkten. Kann nur innerhalb von <#..#> benutzt werden, z.B. in

```
: .Int ( d -- )
  saveSign <# #Int addSign #> type ;
```

#Frac ud1 n -- ud
Erzeugt aus ud1 einen Ausgabestring mit n Nachkommastellen und einem vorrangestellten Komma. Kann nur innerhalb von <#..#> benutzt werden, z.B. in

```
: .Frac ( d n -- )
     >r saveSign
  <# r> #frac  ascii 0 hold addSign #> type ;
```

saveSign d -- sign ud
Bringt das Vorzeichen von d an dritter Stelle auf dem Stack, und nimmt den Absolutwert von d. Dies ist notwendig, weil <#..#> nur mit absoluten Werten arbeiten kann.

addSign n ud -- ud
Addiert zum gerade in Aufbau befindlichen Ausgabestring das Vorzeichen von n. Kann nur innerhalb von <#..#> benutzt werden. ud bleibt dabei unveraendert!

#String d n -- addr n1
Nimmt die doppelt genaue Zahl d mit n Nachkommastellen vom Stack, und hinterlaesst einen String. Der String enthaehlt Tausenderpunkte. Wenn n>0, dann enthaehlt der String ein Dezimalkomma und n Nachkommastellen. Mit n<=0 wird d als ganze Zahl ohne Nachkommastellen interpretiert.

B. Molte, 4920 Lemgo, Steinstoss 24, Tel. 05231/72823

Formatierte Zahlenausgabe in Forth83 mit Dezimalkomma und Tausenderpunkten

```
decimal

: MU/        ( ud un -- ud-q )
 >r  0 r@  um/mod r>  swap >r  um/mod nip  r> ;


: DStellen  ( ud -- n )
 1 begin >r 10 MU/  2dup or while  r> 1+ repeat
   2drop r> ;


: TPunkte  ( ud -- n )
 DStellen  3 /mod  swap 0= + 0 max ;


: #Int      ( ud1 -- ud2 )
 2dup  TPunkte ?dup
  if  0 do # # #  ascii . hold  loop  then #S ;


: #Frac     ( ud1 n -- ud2 )
 abs ?dup
  if  0 do # loop   ascii , hold  then  ;


: addSign   ( n ud -- ud )
 rot sign ;


: saveSign  ( d -- n ud )
  dup -rot  dabs  ;


: #String  ( d n -- addr n )
    >r saveSign
 <# r> #Frac  #Int  addSign #> ;
```

Erstellt im Maerz '87 von:

Bernfried Molte
4920 Lemgo
Steinstoss 24

INPUT# Nachlese

Im den letzten Heften 1986 hatten wir mit kleinen Aufgaben für die Forthprogrammierer Europas begonnen. So sollte in VD 4/86 INPUT# geschrieben werden. Marc Petreman entwickelte in VD 1/87 seine Version mithilfe des EXPECT in Forth-83. In fig-Forth schrieb J. Polster seine Lösung. Er kreiert einen eigenen Input-Buffer, der KEY für KEY ediert wird. Gelungen ist der erweiterte Ansatz, ein Flag zu übergeben, das angibt, ob überhaupt eine Eingabe erfolgte. Hier sein Beispiel.

Diese Worte habe ich schon lange vorher geschrieben, daher die eigene Bufferadresse und die Flag. Da diese Anwendung auf einem Minimalsystem ohne TIB lief - Maschinensteuerung mit Program im Eprom - , legte ich den IBUF an das Ende des in diesem Moment dort nicht benutzten Parameterstacks. Die Flag wird hinterlassen, damit ich dem Programm auch mitteilen kann, dass keine Eingabe stattfindet (ESC), denn 0 ist auch eine Zahl.

Zuerst wird für n+2 Ziffern der Buffer gelöscht, dann die maximale Anzahl Ziffern und die Bufferadresse auf den Maschinenstack und eine 0 als Zähler auf den Parameterstack gelegt, bevor die Tastaturabfrage beginnt.

Bei Eingabe von ESC (27) wird der Zähler vom Stack entfernt, der Maschinenstack geräumt, 0 als Flag auf den Stack gelegt und die Schleife verlassen.

Bei Eingabe von RET (13) wird der Zähler vom Stack auf das erste Byte des Buffers gelegt, durch NUMBER der Eingabestring als doppeltgenaue Zahl auf den Stack gelegt, der Maschinenstack geräumt, 1 als Flag auf den Stack gelegt und die Schleife verlassen.

Bei Eingabe von BS (127 oder ev. 8) wird erst die entsprechende Position im Buffer mit BL überschrieben, dann falls falls der Zähler grösser 0 dieser dekrementiert, BS auf den Bildschirm ausgegeben und die Schleife wiederholt.

Die Eingabe auf Ziffern 0 bis 9, "-" als erstes Zeichen oder Dezimalpunkt geprüfft. Trifft dies zu, wird, falls die rechte Grenze nicht erreicht ist 1, sonst BS an den Monitor ausgegeben und 0 zum Zähler addiert, dann der Wert in die durch den Zähler gezeigte Stelle im Buffer gelegt und auch auf den Bildschirm ausgegeben. Nun wir die Schleife wiederholt.

Bei einer ungültigen Eingabe wird eine akkustische Warnung ausgegeben und die Schleife repetiert.

Das Komma und das Pluszeichen werden nicht akkzeptiert, da das Wort NUMBER mit denselben auch nichts anzufangen weiss.

```
Screen # 5
  Ø ( input )          DECIMAL          ( J. Polster, CH-8820 Waedenswil )
  1 ( eingabe n-ziffern in buffer, ausgabe auf konsole )
  2 ( a n --- Ø      oder              a n --- d 1 )
  3 : INPUT# 2DUP 2+ BLANKS >R >R Ø ( a n --- Ø  oder  a n --- d 1 )
  4   BEGIN KEY CASE                                    ( key-input )
  5     27  OF DROP R> R> 2DROP Ø ;S ENDOF               ( escape )
  6     13  OF R C! R> NUMBER R> DROP 1 ;S ENDOF         ( enter )
  7     127 OF BL OVER R + C! DUP                       ( backspace )
  8            IF 1- BS ENDIF ENDOF
  9       OVER Ø= OVER 45 = AND                         ( sign. minus )
 1Ø       OVER 46 58 INCLUDE OR OVER 47 = Ø= AND        ( ziffer, "." )
 11       IF OVER R' < DUP Ø= IF 8 EMIT ENDIF           ( feld-grenze? )
 12         ROT + 2DUP R + C! SWAP DUP EMIT             ( store + emit )
 13       ELSE BELL ENDIF                               ( error )
 14    ENDCASE AGAIN ;    ( CASE statement by Charles E. Eaker )
 15 -->                   ( from FORTH DIMENSIONS II/3 page 37 )
```

Neues vom forthTREE

von Marco Pauck, Hamburg

Der forthTREE ist die, vor etwa einem Jahr in Betrieb genommene, Mailbox der Forth Gesellschaft eV. Dieser Artikel gibt eine Übersicht über die Änderungen und Neuheiten, die sich seit der letzten Beschreibung des TREE's, hier in der VIERTEn DIMENSION, ergeben haben. Damit möchte ich vor allem jene FORTHler, die immer noch nicht "DFÜ-fähig" sind motivieren, sich vielleicht doch in dieses interessante Gebiet zu wagen. Auf die grundlegenden Prinzipien des TREE's soll hier nicht mehr eingegangen werden. Man kann diese in der VIERTEn DIMENSION Vol.2/No.2 nachlesen. Natürlich findet man auch im TREE selbst alle notwendigen Erklärungen und Hinweise zur Bedienung.

1 Technische Entwicklungen

Die technischen Änderungen, die sich in den vergangenen Monaten aufgrund der Erfahrungen mit dem Betrieb ergeben haben, dienten hauptsächlich dazu, den Umgang mit dem TREE zu erleichtern.

Als erstes wurde die gesamte Benutzeroberfläche eingedeutscht. Die beiden wichtigsten Befehle heißen nun LESE und INDEX. Es reicht aus sie zu kennen, um sich einen ersten Eindruck vom TREE zu verschaffen und etwas in ihm herumzustöbern. Eine weitere, wesentliche Verbesserung der Benutzeroberfläche ist die Möglichkeit, nunmehr neben den üblichen Befehlen auch die Namen der Nachrichten abkürzen zu können. Ich denke, daß der TREE mit diesen beiden Änderungen jetzt wirklich komfortabel und verständlich in der Benutzung ist.

2 Inhaltliche Entwicklungen

Was tut sich momentan inhaltlich im TREE? Stellen wir uns einfach vor, wir wären bereits "eingeloggt". Befehle und Nachrichtennamen können, wie bereits erwähnt, auch abgekürzt werden. Der Lesbarkeit halber sind sie hier jedoch ausgeschrieben:

```
Befehl? Lese Konferenzen

*** 09-MAE-86  KONFERENZEN
gelesen:2315

z.Z. laufen folgende Konferenzen:

Folgenachrichten:
.02-JUN-86  FORTH
.11-APR-86  GRAFFITI
.11-MAI-86  KOMMUNIKATION
.16-MAI-86  UMWELT
.18-JUL-86  POLITIK
.01-JUN-86  RECHNERECKE
.16-APR-86  GAST
.12-NOV-86  PRIVAT
.05-AUG-86  SYS-INFO
.24-MAE-87  EINFUEHRUNG
.17-APR-87  HILFE
```

Der Hauptzweig FORTH interessiert uns natürlich am meisten. Er ist in der Zwischenzeit beständig gewachsen und unterteilt sich in folgende Unterzweige:

```
Befehl? Lese Forth

*** 02-JUN-86  FORTH
Vorgaenger: KONFERENZEN
gelesen: 568

Hier erfaehrt man alles ueber Forth und die Forth Gesellschaft eV.
Hier kann man auch Fragen zu allgemeinen und spezielleren Forthproblemen
stellen und Forth-Software finden.

Folgenachrichten:
.14-DEZ-86  DIE.GESELLSCHAFT
.07-MAI-86  SYSTEME
.01-JUN-86  MASCHINEN+CHIPS
.07-MAI-86  ASK.THE.DOCTOR
.31-AUG-86  PROGRAMME
.05-FEB-87  DIVERSES
```

Im Zweig DIE.GESELLSCHAFT finden sich für Außenstehende allgemeine Informationen über die Forth Gesellschaft eV, sowie für Mitglieder Aktualitäten aus den Lokalen Gruppen und Fachgruppen.

Der Zweig SYSTEME enthält Informationen über die verschiedene Forth-Systeme. Vor allem volksFORTH, figFORTH und das F83 sind dort in Form von Update-Neuigkeiten, Bug-Reports u.ä. vertreten.

MASCHINEN+CHIPS enthält "brandheiße" Neuigkeiten, Erfahrungsberichte und Tips zu allen derzeit existierenden Forth-Maschinen und Forth-Chips. Der Novix NC4000 ist besonders stark vertreten. Der forthTREE dürfte, bundesweit gesehen, die umfassenste Informationsquelle zu diesem Thema sein.

In ASK.THE.DOCTOR können Fragen zu Forth-Problemen gestellt werden. Unter dem Motto: "Alles, was Sie schon immer über Forth wissen wollten, sich aber nicht zu fragen trauten". Meist findet sich auch auf ausgefallene Fragen schnell eine Antwort.

Forth-Software befindet sich im Zweig PROGRAMME. Besondere Leckerbissen dürften das cmFORTH von Chuck Moore, das LISP von Ulrich Hoffmann und ein TREE-ähnliches Mailbox-Programm sein. Neben recht langen Programmen sind auch viele "Ein-Screener" vorhanden.

Was nicht recht in die o.g. Zweige paßt, findet sich dann noch im Zweig DIVERSES. Dort hängen momentan z.B. aktuelle Hinweise auf VHS-Kurse über Forth.

3 Wer sind die Benutzer des TREE's?

Der TREE ist *die* Informationsquelle für alle Forth-Interessenten. Aber außer dem Angebot von Informationen, wie es die VIERTE DIMENSION auch bietet, ermöglicht es der TREE Kontaktsuchenden, *selbst* entsprechende Gesuche oder Aufrufe in den TREE zu hängen. Es ist kein Passwort oder eine sonstige Zugangsberechtigung notwendig, um eigene Nachrichten an beliebige Zweige zu hängen. Deshalb, und aufgrund der Baumstruktur, bietet sich der TREE vor allem für Diskussionen an, wie sie in einer Zeitschrift nicht geführt werden können.

Der Hauptzweig FORTH ist nur einer von mehreren. Die anderen Zweige, wie UMWELT oder POLITIK werden von der hiesigen "Mailbox-Szene" mitgenutzt. Auf diese Weise macht der TREE zum einen Öffentlichkeitsarbeit und enthält zum anderen eine recht bunte Mischung von Informationen und Meinungen.

Der forthTREE ist 24 Stunden täglich mit 300bd/8N1 zu erreichen unter:

Tel.: 040-3904204

Quelltextdokumentation

Bernd Pennemann, Hamburg

Es wird ein kurzes Programm vorgestellt, das die Erzeugung einer Dokumentation zu Forthprogrammen erleichtert. Es generiert aus dem Quelltext des Programms einen Text, der zur Herstellung eines Glossars dienen kann.

Schlüsselworte : volksFORTH83, Quelltextdokumentation, CREATE, Stil

Beispiel in Bild 1 zeigt, was das Programm leistet. Es erzeugt aus einem Quelltext eine Art Glossar, in das der Textteil eingefügt werden muß. Das Programm entstand, als die GEM-Bibliothek des volksFORTH83 bei der grundlegenden Überarbeitung des Handbuchs dokumentiert werden sollte. Das Problem bestand darin, alle Namen korrekt (!) abzutippen, die Stackkommentare möglichst fehlerfrei zu übertragen und dem Leser auf einen Blick mitzuteilen, um was für eine Art von Wort es sich handelt.

Das Programm setzt eine bestimmte Struktur des Quelltextes voraus. Da Forth
Das Format des Quelltextes sieht folgendermaßen aus :

Dem Namen des zu erzeugenden Forthwortes ist das definierende Wort vorangestellt. (Wenn das definierende Wort in einer Schleife auftritt (siehe z.B. [2]), funktioniert unser Programm nicht korrekt.)

Auf den Namen folgt ein Kommentar in Klammern, der die Stackwerte in der üblichen Notation vor und nach der Ausführung des Wortes darstellt (siehe z.B. [3-5]). Der Stackkommentar darf fehlen.

Auf den Stackkommentar folgt ein höchstens einzeiliger Kommentar, der durch \ angeführt wird. Dieser Kommentar kann ebenfalls weggelassen werden.

Die Quelltexte des volksFORTH83 besitzen weitgehend diese Struktur, ebenso das Listing in [1].
Das unten abgedruckte Listing ist für das volksFORTH83 geschrieben, mit einigen Kenntnissen kann es jedoch auch an andere Forthsysteme angepaßt werden. Einige der sehr volksFORTH83-spezifischen Worte sind auf Screen 2 knapp erklärt. Ausführlichere Erläuterungen finden Sie in [1].

Beachten Sie bitte, daß CREATE das einzige Wort sein muß, das Namen im Dictionary des volksFORTH erzeugt. Nur so ist sichergestellt, daß wirklich alle definierten Worte dokumentiert werden, denn in die erste Position von CREATE wird unser Wort PRINTHEADER gepatched. Der Inhalt dieser Position wird nun wieder an das Ende von PRINTHEADER gepatched, sodaß die Funktion von CREATE nicht beeinträchtigt wird. Bei Ausführung von PRINTHEADER zeigt >IN auf den Namen des zu definierenden Wortes, denn CREATE liest diesen Namen ein und PRINTHEADER ist nun das erste Wort in CREATE .

Noch ein paar Tips für volksFORTH-Benutzer :

Die Zeile "LISTWOR2.SCR Blk 1" im Beispiel wird unterdrückt, falls man **' noop Is .status** eingibt.

Will man die Warnung " exists " unterdrücken, die ausgegeben wird, falls ein Wort redefiniert wird, so gebe man **1 warning !** ein.

Außerdem kann man den Ausdruck formatieren, z.B. indem man den Stackkommentar immer ab Spalte 30 ausdruckt. Dazu verwende man die Worte AT? und AT .

Fort(h)geschrittene Programmierer sind sicherlich auch in der Lage, die Ausgabe nicht nur auf den Bildschirm oder auf den Drucker umzuleiten, sondern auch in ein File, das dann vielleicht gleichzeitig einen Index für den erklärenden Text des Glossars darstellt....

[1] B.Pennemann, Ein Terminalprogramm in Forth, Vierte Dimension II/2, S. 20-32

[2] K.Scheller, "Defining Words, eine Einführung in die Anwendung", Vierte Dimension II/3, S. 19-20

[3] Forth Standards Team, Forth-83 Standard, (c) 1983 Mountain View USA

[4] L. Brodie, Style Conventions, Nachdruck in Vierte Dimension II/2, S. 8-12

[5] Kim Harris, Forth Coding conventions, 1985 FORML Conferences proceedings, S. 143-186

```
LISTWOR2.SCR  Blk 1

:  >adr  ( -- adr )
:  .definer  ( -- ) \ geht nicht bei  DO..CREATE..LOOP
:  .text  ( char -- )
:  printheader  ( -- )
 : +listwords
 : -listwords
```

Bild 1

0

```
0               ## LISTWOR2.SCR ##                    bp 26dec86
1
2 Dieses File gibt nach Ausführen von +LISTWORDS bei jedem
3  kompilierten Wort das zugehörige definierende Wort, das
4  Wort selbst und soweit vorhanden, einen Stackkommentar
5  und eine Kommentarzeile aus.
6
7 Mit -LISTWORDS wird diese Funktion wieder abgeschaltet.
8
9
10
11
12
13
14
15
```

1

```
0 \ Drucke alle Definitionen aus                      bp 26dec86
1
2 : )adr        ( -- adr )  source  drop  )in @ + ;
3
4 : .definer    ( -- )      \ geht nicht bei  DO..CREATE..LOOP
5   )adr 1- dup BEGIN  1- dup c@ bl = UNTIL  under - type space ;
6
7 : .text       ( char -- )  dup word count type emit ;
8 : printheader ( -- )  blk @ 0= ?exit      )in push   cr
9   ?head @  IF  .' :'  THEN   .definer  bl .text   name  @
10  ' (' @  case?  IF   .' ('   Ascii ) .text   name @ THEN
11  ' \' @  =      IF   .' \'   )adr c/l  )in @ c/l mod  -
12                      type  THEN   [ ' Create )body @ , ] ;
13
14 : +listwords   ['] printheader ['] Create )body ! ;
15 : -listwords   [ ' Create )body dup @ ] Literal Literal ! ;
```

2

```
0               Systemspezfisches :
1   hex  20 Constant bl
2
3   : source        ( -- adr len )
4         blk @ ?dup  IF  block  $400  ELSE  tib #tib @ ;
5
6   : name          ( -- adr )      bl word ;
7   : case?         ( n1 n2 -- n1 ff / tf )
8         over = dup  IF  swap drop THEN
9
10  )IN PUSH   kann durch  )IN @ )R  sowie  R) )IN !  vor dem ;
11        ersetzt werden.
12
13  ?HEAD @  IF  .' :'  THEN  entfällt auf anderen Systemen
14
15  ?EXIT    kann durch  IF EXIT THEN  ersetzt werden.
```

3

```
)ADR            adr zeigt auf den Namen des zu definierenden
                  Wortes im Quelltext.
.DEFINER        druckt den letzten im Quelltext gefundenen
                  Namen aus; der sollte das definierende Wort sein.

.TEXT           druckt den Quelltext bis zum (einschließlich)
                  nächsten Auftreten des Zeichens c aus.
PRINTHEADER     druckt das definierende Wort, den Namen des
                  Wortes sowie einen Balken für Namen auf dem Heap und folgende
                  Kommentare aus. Außerdem tut's das, was in CREATE vor dem
                  patchen stand.

+LISTWORDS      patched CREATE
-LISTWORDS      patched CREATE zurück.
```

Und sie müssen zwei Köpfe haben und Feuer spucken können...

Mitarbeiter gesucht

Das FORTH MAGAZIN sucht Mitarbeiter für die Redaktion.

Sie müssen können:

Programmieren in Forth, recherchieren, redigieren, englisch, Maschinen schreiben. Voraussetzung sind eigene Textverarbeitung mit DFÜ. Sie arbeiten selbständig, gewissenhaft, zuverlässig und prompt an Ihren Aufgaben.

Interessenten wenden sich an:

Michael Kalus, Tel: 02336-82204.

Laufzeitmessung von Forthworten

Bernd Pennemann, Hamburg

In dem folgenden Artikel wird ein Hilfsmittel zur Bestimmung der Aufrufhäufigkeit und der Ausführungszeit von Forthworten vorgestellt. Der Code ist für das volksFORTH83 auf dem Atari ST abgedruckt, kann aber sehr leicht auf andere Rechner angepaßt werden. Auch für andere Forthsysteme sollte eine Anpassung leicht möglich sein.

Schlüsselworte : Aufrufhäufigkeit, Ausführungszeit, Atari ST, volksFORTH83

Es wird oft empfohlen, Programme zunächst in Hochsprachen zu verfassen und zeitkritische Teile des Codes erst gegen Ende der Entwicklung in Maschinensprache umzuschreiben. Diese Vorgehensweise hat den Vorteil, daß der dem Programm zugrundeliegende Algorithmus einfacher getestet werden kann, als wenn das Programm sofort und gänzlich in Maschinencode geschrieben würde. Außerdem wird die Übertragbarkeit des Programms auf andere Rechner durch die Verwendung einer Hochsprache deutlich gesteigert und das vollständig in Hochsprache entwickelte Programm kann zu Dokumentationszwecken weiterverwendet werden. Schließlich kann oft erst anhand des fertig entwickelten Programms festgestellt werden, welche Routinen "zeitkritisch" sind und in Maschinensprache übersetzt werden müssen. Als Faustformel wird oft angegeben, daß ein Programm 90 % seiner Rechenzeit in 10 % des Programmtextes verbringt. Das bedeutet, daß für eine wesentliche Verringerung der Rechenzeit oft nur ein sehr kleiner Teil des Programms in Maschinensprache übersetzt werden muß. Das Problem besteht nun darin, herauszufinden, welcher Teil des Programmtextes das ist.
Hierzu gibt es sowohl theoretische [1] als auch praktische Vorschläge [2]. Im vorliegenden Artikel wird ein Proramm vorgestellt, das ähnlich wie [2] arbeitet. Das hier vorgestellte Programm hat jedoch eine Reihe von Vorteilen : Es läuft auf dem volksFORTH83, die Meßfehler sind geringer und es ist dem Autor verständlich.

Das vorgestellte Programm benötigt eine Speicherstelle im Rechner, deren Inhalt in periodischen Abständen erhöht wird. Diese Speicherstelle wird im Listing mit TIMER bezeichnet. Ist so eine Speicherstelle nicht vorhanden, so kann evtl. ein im Rechner eingebauter Timerbaustein dazu veranlaßt werden, in regelmäßigen Abständen einen Interrupt auszulösen. Die Interruptroutine zählt dann eine vorgegeben Speicherstelle hoch. Die Frequenz, mit der die Erhöhung stattfindet, sollte größer als 50 Hz sein, damit eine befriedigende Auflösung der Rechenzeiten erzielt wird. Beim C64 ist diese Adresse übrigens $A1; sie wird 60-mal pro Sekunde erhöht und eignet sich damit gut für unsere Zwecke.

Gebrauchsanleitung

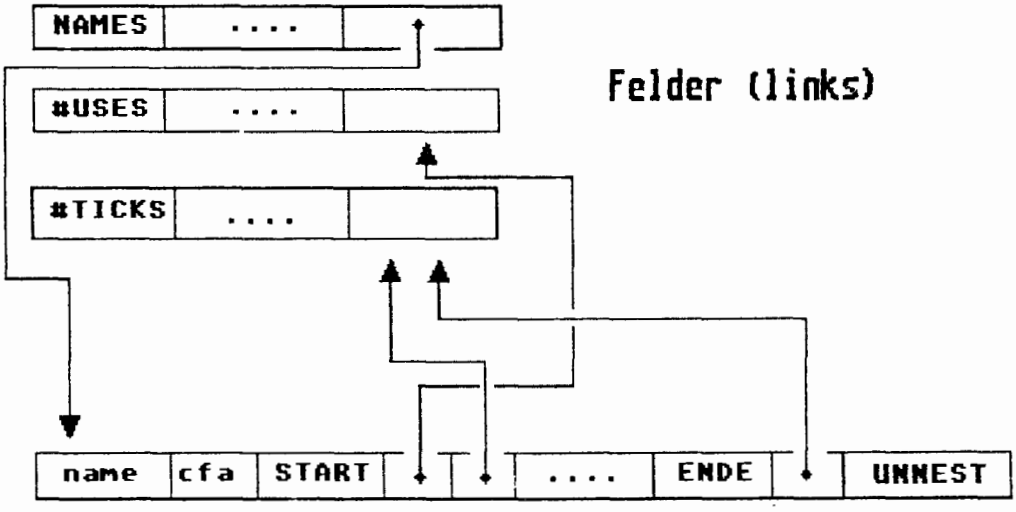
Nach dem Laden der im Listing abgedruckten Screens können alle anschließend kompilierten :-Definition analysiert werden. Mit Codeworten und Worten, die vor dem Laden dieser Screens definiert wurden, funktioniert das ganze nicht !
Man gibt zunächst NEW ein, um die Ausführungszeiten zu löschen und startet anschließend das zu analysierende Programm. Nachdem die Ausführung beendet wurde, erhält man mit .TABLE eine Tabelle, wie sie im Listing für ein Beispielprogramm abgedruckt ist. Besonders vorteilhaft ist dabei, das die Ausführungszeit aller benutzen Worte simultan bestimmt werden kann.
So benötigt z.B. das Wort DUP+ des Beispielprogramms (Screen 10) auf dem Atari ST ca. 24,5 Mikrosekunden (der *gemessene* Wert schwankt etwas) Ausführungszeit (siehe Tabelle).
Man kann nun mehrere Messungen akkumulieren oder eine neue Messung mit NEW einleiten.

Wie funktioniert's ?

Es werden zunächst Tabellen angelegt, die für jedes zu untersuchende Wort Platz für die Speicherung der Zahl der Aufrufe, der dafür benötigen Systemtakte und für einen Zeiger auf den Namen des Wortes zur Verfügung stellen. Der Zeiger auf den Namen wird für die Ausgabe der Tabelle benötigt. Die Zahl der (bisherigen) Einträge in die Tabelle wird durch die Variable #ENTRIES angegeben, die maximal zulässige Anzahl durch die Konstante #WORDS . Sie ist auf 127 Worte begrenzt, kann aber natürlich verändert werden. Anschließend werden einige Worte zur formatierten Ausgabe der Daten definiert.

Der eigentliche Trick bei diesem Programm befindet sich auf den Screens 7 bis 9. Auf Screen 9 werden nämlich die Worte : und ; (bzw. EXIT) modifiziert, so daß sie zusätzlichen Code in jedes neu definierte Wort kompilieren, der dafür sorgt, daß bei jeder Ausführung des Wortes die Zahl der Aufrufe um eins erhöht und die dafür benötigte Zeit bestimmt wird. Zu diesem Zweck wird an den Anfang des Wortes die Routine START, zusammen mit Zeigern auf die Felder #USES und TICKS, eingetragen.

START holt sich nun durch geschickte Manipulation am Returnstack diese beiden Zeiger. Der Code funktioniert übrigens nur bei "indirect threaded code"-Systemen, die über einen "post-incrementing"-Adreßinterpreter verfügen. In unserem Zusammenhang ist nur relevant, daß in diesem Fall innerhalb einer :-Definition die Adresse des *nächsten auszuführenden* Wortes innerhalb des *aufrufenden* Wortes auf dem *Returnstack* liegt.



Struktur eines zu stoppenden Wortes (oben)

Im konkreten Fall findet START auf dem Returnstack also die Adresse des Zeigers auf ein Element des Arrays #USES. Das Bild verdeutlicht diese Zusammenhänge.

Die meisten Forthsysteme (volksFORTH83, figFORTH und F83) erfüllen diese Bedingung, so daß man sich in der Regel darüber keine Gedanken zu machen braucht.

Nun tritt allerdings ein heikles Problem auf. Die Worte START und ENDE benötigen selbst recht viel Rechenzeit, die die gemessene Gesamtzeit stark erhöhen kann. Das gilt insbesondere für Worte, die gestoppt werden und selbst viele Worte benutzen, die ebenfalls gestoppt werden. In diesem Fall können die gemessenen Ausführungszeiten völlig falsch sein! Abhilfe kann nur durch Optimierung dieser beiden Worte geschaffen werden; sie müssen unbedingt in Code geschrieben werden. Auf Screen 8 ist der Code für den 68000-Assembler vorgestellt worden. Hier wurde aus Geschwindigkeitsgründen noch eine weitere Optimierung vorgenommen: Die Umrechnung der Adressen des Forthsystems in

absolute Speicheradressen geschieht bereits während der Kompilation durch das Wort L, ! Auf anderen Systemen ist das natürlich nicht erforderlich. Aber auch wenn START und ENDE in Code geschrieben werden, können sie ziemlich viel Rechenzeit benötigen. Das Beispiel auf Screen 10 zeigt so ziemlich den schlechtesten Fall, denn innerhalb der Redefinition des Wortes 2* benötigt das alte, in Code geschriebene Wort 2* nur sehr wenig Zeit, so daß die Worte START und ENDE stark zu Buche schlagen.
Die Tabelle zeigt, daß das Wort TT2* in diesem Fall genau 2.105 Sekunden benötigt. Ersetzt man : und ; bei den Worten 2* DUP+ und TWO* durch OLD: und ;OLD , so werden sie ohne START und ENDE kompiliert; es wird also nur noch T2* und TT2* gestoppt. In diesem Fall benötigt TT2* nur noch 1.430 Sekunden Rechenzeit. Im abgedruckten Beispiel, das, wie erwähnt, einen sehr ungünstigen Fall darstellt, wird also 30 % der ursprünglich gemessenen Rechenzeit in den Worten START und ENDE verbracht.
Nun mag sich mancher Leser darüber wundern, daß es gelingt, die Ausführungszeiten von Worten zu messen, die erheblich kleiner als der zeitliche Abstand ist, mit der die Systemuhr hochgezählt wird. Das kann man jedoch verstehen, wenn man sich überlegt, daß die *Wahrscheinlichkeit*, daß die Uhr während der Ausführung eines Wortes erhöht wird, umso größer ist, je größer die *gesamte* Ausführungszeit des Wortes wird.
Der praktische Nutzen dieses Programms ist bei größeren Anwendungen beträchtlich; steht man vor dem Problem der Optimierung eines Programms, so treten die zeitkritischen Stellen oft unerwartet auf.

Dank an Ulrich Hoffmann; er hat mich auf die Idee mit dem Balkendiagramm gebracht.

[1] "Performance Analysis in Threaded Code Systems", Michael Perry, nachgedruckt in Vierte Dimension, Vol.II, Nr.3, p.23-26

[2] "Forth Timer Macros", Iram Weinstein, Forth Dimensions, Vol.7, No.3, p. 19-26

Zur Laufzeitmessung von Forthworten

B. Pennemann geht davon aus, daß der Zeitpunkt der Ausführung eines beliebigen Wortes in einer Applikation in keinem festen Verhältnis zum Systemtakt steht. Unter dieser Voraussetzung ist seine Art der Laufzeitabschätzung aussagefähig. In Echtzeitanwendungen ist diese Methode jedoch nicht oder nur schlecht zu gebrauchen, denn diese beinhalten in der Regel Interrupts zu festen Zeiten relativ zur Systemuhr. Die Laufzeitmessung solcher Worte braucht dann Timerbausteine. (mk)

volksFORTH-83 FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks TIMING.SCR Seite 1

0

```
0 \ TIMING.SCR
1
2 Dieses File enthält Worte, die eine Tabelle der Ausführungs-
3  zeiten für eine Applikation herstellen.
4
5 Alle :-Definitionen werden in einer Tabelle mit der Zahl
6  der Aufrufe und der gesamten benötigten Rechenzeit
7  ausgegeben.
8
9 NEW            Initialisiert die Tabelle
10 .TABLE        Gibt eine Tabelle aus.
11
12
13
14
15
```

1

```
0 \ loadscreen                                  bp 12may87 %
1
2 Onlyforth
3
4 \needs Code     2 loadfrom assemble.scr                    lade den Assembler, falls er noch nicht vorhanden ist.
5
6 1 9 +thru
7
8
9
10
11
12
13
14
15
```

2

```
0 \ Uhr für volksFORTH83 auf dem Atari          bp27apr87 %
1
2 ( $4BC Constant timer                                TIMER          Adresse der Speicherzelle, die mit 200 Hz
3                                                                     hochgezählt wird. In Wirklichkeit handelt es
4                                                                     sich um eine doppelt genaue Variable, aber
5                                                                     wir brauchen nur die unteren 16 Bit.
6
7 ( : gettimer    ( -- n )                             GETTIMER       liefert den Inhalt des Timers auf dem Stack.
8     timer 0 l@ ;
9
10
11 : tinit ;                                           TINIT          initialisiert den Timer. Hier benutzen wir den
12                                                                    Systemtimer, so daß nichts weiter getan werden
13                                                                    muß.
14
15 &200 Constant ticks/sec                             TICKS/SEC      wie der Name sagt...
```

11

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

12

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

13

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

volksFORTH-83 FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks TIMING.SCR Seite 2

3 14

```
0 \ primitives for adding time                          bp 28mar87 %
1
2 : Array   Create   2* allot     Does)  swap 2* + ;
3
4 $7F Constant #words
5
6 Variable #entries              #entries off
7
8 #words Array #uses             \ Number of calls of a word
9 #words Array #ticks            \ total time consumed
10 #words Array names            \ ptr. to name of word
11
12 : new                                                  NEW
13    #words 2*  0 #uses over erase   0 #ticks swap erase  tinit ;
14
15 new
```

ARRAY	ist ein definierendes Wort für Arrays.
#WORDS	Größe der unten definierten Tabelle. Es können max. #WORDS Worte gestoppt werden.
#ENTRIES	Zahl der Worte, die gestoppt werden sollen.
#USES	Zahl der Aufrufe eines Wortes.
#TICKS	Zahl der Systemtakte, die dafür benötigt wurden.
NAMES	Zeiger auf die Namen der gestoppten Worte.
NEW	Bereite eine Messung vor.

4 15

```
0 \ print a table of execution time                     bp 12may87 %                                    bp 12may87
1
2 : tab          ( n -- )  col - 1 max  spaces ;                    TAB
3                                                                   COL liefert die momentane Position des Cursors,
4 : .header
5    cr cr 15 tab  ." volksFORTH83 statistic analysis"              .HEADER
6       cr 15 tab  ." ============ ========= ========"              gibt den Kopf der Tabelle aus.
7    cr cr          ." Name "     &15 tab ." #Uses"                 Dazu gehört eine Überschrift und 4 Spalten.
8          &30 tab ." Total"     &40 tab ."  Bar diagram"
9       cr  &30 tab ." [sec]"  cr ;
10
11 : u*/-d       ( u1 u2 u3 -- udqout )                             U*/-D
12    )r  um*  r)  ud/mod  rot drop ;                               u1*u2/u3 liefert den doppelt genauen Quotienten
13                                                                  udquot. Dieses Wort erfüllt die Funktion von
14                                                                  */ , jedoch arbeitet es mit vorzeichenlosen
15                                                                  Zahlen und liefert ein doppelt genaues Ergebnis.
```

5 16

```
0 \ "                                                   bp 12may87 %
1
2 : msec         ( u -- ud )     &1000 ticks/sec  u*/-d ;           MSEC
3                                                                   Millisekunden.
4 : .msec        ( ud n -- )     \ drucke rechtsbündig
5     -rot  (#  # # #  Ascii . hold  #S #)                         .MSEC
6     rot over max   over - spaces type ;                           "Nachkommastellen" aus.
7
8 : getmaxtime    ( -- u )                                        GETMAXTIME
9    0  #entries @ 0 ?DO  I #ticks @ umax  LOOP ;                   Auf diesen Wert wird das Balkendiagramm
10                                                                  skaliert.
11 &35 Constant maxbar                                            MAXBAR
12                                                                  auf dem Bildschirm angibt.
13 : drawbar     ( umax u -- )                                    DRAWBAR
14    maxbar rot  u*/-d  drop                                       den Wert umax.
15    0 ?DO  Ascii # emit  LOOP  ;
```

MSEC	konvertiert die Zahl der Systemtakte u in Millisekunden.
.MSEC	gibt eine doppelt genaue Zahl mit drei "Nachkommastellen" aus.
GETMAXTIME	liefert das Maximum der Ausführungszeiten. Auf diesen Wert wird das Balkendiagramm skaliert.
MAXBAR	Eine Konstante, die die max. Länge der Balken auf dem Bildschirm angibt.
DRAWBAR	zeichnet einen Balken der Länge u , skaliert auf den Wert umax.

volksFORTH-83 FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks TIMING.SCR Seite 3

6

```
0 \ *                                                  bp 12may87 %
1
2 : .entry      ( max  n -- )   base push  decimal
3     dup #uses  @  0= IF  2drop exit  THEN     cr
4               dup names  @  .name
5     &15 tab   dup #uses  @  5 u.r
6     &28 tab   dup #ticks @  msec 7 .msec
7     &40 tab       #ticks @  drawbar ;
8
9 : .table
10    .header    getmaxtime   #entries @  0
11    ?DO  dup  I .entry  stop?  IF  LEAVE  THEN  LOOP
12    drop  cr ;
13
14
15
```

17

.ENTRY gibt Name, Zahl der Aufrufe und Rechenzeit des n-ten Wortes in den obigen Arrays aus. Schließlich malt es noch ein Balkendiagramm.

.TABLE gibt alle Daten in Tabellenform aus. Die Ausgabe kann in der Üblichen Weise angehalten und abgebrochen werden.

7

```
0 \\ words for adding ticks                            bp 2oct86 %
1
2 : start       ( -- )
3     gettimer
4          1  r) dup  2+ )r  @ +!     \ hole IP vom Returnstack
5     negate  r) dup  2+ )r  @ +!  ;
6
7 : ende        ( -- )
8     gettimer  r) dup  2+ )r  @ +!  ;
9
10
11
12 : l,          ( adr -- ) , ;
13
14
15
```

18

Diese Worte sind zu rein illustrativen Zwecken vorgestellt. Bei realen Anwendungen müssen sie in Code geschrieben werden; möglichst schnell, so daß der Meßfehler gering wird !

FREEZE speichert die Uhr, so daß START und ENDE selbst weniger Rechenzeit bewirken.

START erhöht die Zahl der Aufrufe und merkt sich den Uhrstand. Auf dem Returnstack befindet sich der IP, der auf das nächste auszuführende Wort zeigt...

ENDE addiert die verstrichene Zeit zu dem Tabellenwert hinzu. Da wir schon den Anfangswert in START abgezogen haben, werden nur die verstrichenen Ticks berücksichtigt.

L, kompiliert die Adresse eines Arrayelements.

8

```
0 \ words for adding ticks in code   FAST              bp27apr87 %
1
2 Code start      ( -- )
3     .l IP )+ A0 move          \ Hole Zeiger auf #USES
4     .w 1 A0 ) addq            \ erhöhe #USES
5     .l IP )+ A0 move          \ Hole Zeiger auf #TICKS
6     .w timer #) D1 move       \ subtrahiere (!) Timer
7        D1 A0 ) sub            \ von #TICKS
8     Next   end-code
9
10 Code ende      ( -- )
11    .l IP )+ A0 move          \ Hole Zeiger auf #TICKS
12    .w timer #) D1 move       \ addiere Timer zu #TICKS
13       D1 A0 ) add
14    Next end-code
15 : l,          ( adr -- )  )absaddr , , ;
```

19

Diese Worte entsprechen nicht ganz den obigen Worten. Das volksFORTH auf dem Atari hat die Eigenschaft, irgendwo im Speicher sitzen zu können, ohne daß der Benutzer davon etwas merkt. Er merkt nur etwas, wenn er in Code programmiert. Dann muß er nämlich den Anfang des Forth zu allen Adressen hinzuaddieren. Aus Geschwindigkeitsgründen machen wir das nicht in START bzw. ENDE, sondern bereits während der Compilation in L, . Dann sind die Zeiger in die Tabellen allerdings doppelt genau !

L, Kompiliere eine Adresse als doppelt genaue absolute Adresse in den Speicher.
ACHTUNG : Dieser Code kann NICHT mit SAVESYSTEM abgespeichert werden, was aber in der Regel nicht stört.

```
volksFORTH-83     FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks  TIMING.SCR  Seite 4

                9                                              20

0 \ change compiler for code generation              bp 14may87 %                                    bp 14may87
1
2 ( : newentry      last @   #entries @ names !                   NEWENTRY       trage das gerade definierte Wort in die Tabellen
3     1 #entries +!  #entries @  #words =  abort" table full !" ;               ein. Prüfe, ob die Tabellen voll sind !
4
5 : old:            :  ;                    \ merke die alten :   OLD:           Mit  OLD: ... ;OLD  statt  : ... ; kann
6 : ;old  [compile] ;  ;  immediate restrict \ und ;               ein Wort von der Messung ausgeschlossen werden!
7
8 old: :     : compile start  #entries @ dup #uses 1,  #ticks 1, ; :            wird definiert, so daß neu erzeugte Worte in die
9                                                                  Timertabellen eingetragen werden. Außerdem wird
10 old: exit    compile ende  #entries @  #ticks 1,                der Uhrstand bei Aufruf des Wortes gemerkt.
11              compile exit ;    immediate restrict               EXIT          wird so definiert, daß die Uhr beim Verlassen
12                                                                 des Wortes abgelesen wird.
13
14 old: ;       compile ende  #entries @  #ticks 1,  newentry      ;             wie EXIT ...
15         [compile] ;   ;   immediate restrict

                10                                             21

0 \ test                                             bp 28mar87  %
1
2 : 2*       2* ;                                                  Kleines Testprogramm
3 : dup+     dup + ;
4 : two*     2 * ;
5
6 : t2*         &234   &1000 0 DO
7                       dup 2*    drop
8                       dup dup+  drop
9                       dup two*  drop
10             LOOP  drop ;
11
12 : tt2*   &10 0 DO  t2*  LOOP  ;
13
14 new tt2* .table
15
```

```
                volksFORTH83 statistic analysis
                ============ ========= ========

Name            #Uses          Total       Bar diagram
                               [sec]

2*              10000          0.165       ##
DUP+            10000          0.245       ####
TWO*            10000          0.305       #####
T2*                10          2.100       ####################################
TT2*                1          2.105       ####################################

Tabelle : Ausführungszeiten der Worte des Beispiels
```

Strukturierte Datentypen

Ulrich Hoffmann März 1987

Der folgende Artikel zeigt eine einfache Form der Implementation von strukturierten Datentypen inklusive Forth-83 Source-Code.

Strukturierte Datentypen (Struct in C, Record in Pascal und Modula, etc.) sind ein wesentliches Merkmal für eine "höhere" Programmiersprache.
Forth, dem diese Strukturen von Haus aus fehlen wird mit diesem Argument häufig eine geringe Mächtigkeit bescheinigt.
Um so interessanter ist es festzustellen, daß es sich hierbei wieder ein mal um ein Problem handelt, das in die "Kann auf einem Screen gelöst werden" - Klasse fällt.

Grundlegende Eigenschaft eines Datentyps ist die Größe des Speicherplatzes, den Variablen dieses Typs belegen. Die hier vorgestellten Forth Datentypen geben also lediglich an, wieviel Speicherplatz reserviert werden soll. Mit Hilfe eines weiteren Wortes "Var" werden dann die Variablen der entsprechenden Größe erzeugt.
Dazu ein Beispiel:

Der Datentyp char belegt ein Byte im Speicher:

```
    1 Constant char      ( Definition des Datentyps char )
char Var zeichen         ( Erzeugen einer char-Variablen 1 Byte )
```

Der Datentyp integer belegt zwei Bytes im Speicher:

```
      2 Constant integer ( Definition des Datentyps integer )
integer Var zahl         ( Erzeugen einer integer-Variablen 2 Bytes )
```

Strukturen können aus elementaren Datentypen oder schon bestehenden Strukturen erzeugt werden. Die einzelnen Teile einer Struktur werden dabei über sogenannte Tag-Fields angesprochen. Sie werden mit Hilfe eines speziellen definierenden Wortes erzeugt ("Field").
Auch hier ein Beispiel:

Definiert wird die Struktur Datum:

```
{  integer Field tag    ( Definition Tag-Field tag   2 Bytes )
   integer Field monat  ( Definition Tag-Field monat 2 Bytes )
   integer Field jahr   ( Definition Tag-Field jahr  2 Bytes )
} datum                 ( Definition Datentyp  datum 6 Bytes )

datum Var heute         ( Erzeugen einer Datums-Variablen  6 Bytes )
```

Als dann können die einzelnen Komponenten von "heute" angesprochen werden, also "heute tag", "heute monat" und "heute jahr".
Der neu definierte Datentyp ("datum") gibt bei Ausführung ebenfalls die Größe des Speicherplatzes an, den Variablen dieses Typs belegen (6 Bytes). Er paßt sich damit in das bisherige Schema ein, es ist daher möglich Strukturen von Strukturen zu bilden, etwa:

```
{     8 Field vorname          ( Tag-field    8 Bytes )
      8 Field nachname         ( Tag-field    8 Bytes )
  datum Field geburtsdatum     ( Tag-field vom Typ Datum 6 Bytes )
  datum Field taufdatum        ( Tag-field vom Typ Datum 6 Bytes )
} person                       ( Datentyp "person" 28 Bytes )
```

Neben dem Bilden von Strukturen ist es ebenfalls sehr nützlich viele Variablen gleichen Typs unter einem Typ zusammenfassen zu können. Der Ruf nach Feldern (Arrays, Vektoren, Reihen, ...) wird laut:
Zunächst die Definition von Feldern von Strukturen:

```
person [ 20 ]_Var personenliste  ( Personen Feld mit 20 * 28 Bytes )
```

Die einzelnen Komponenten der Personenliste (diese sind vom Typ Person), lassen sich mit Hilfe eines Index von 0 bis 19 ansprechen, etwa:

```
5 personenliste
```

Von dieser Person ausgehend können nun ebenfalls die Komponenten einer Person angewählt werden, so daß Ausdrücke wie:

```
1987 3 personenliste geburtsdatum jahr !  ( und )
2 personenliste name  20 expect
```

den gewünschten Effekt haben.

Bei der Definiton von Strukturen von Feldern, treten Felder als Komponenten einer Struktur auf. Es muß also möglich sein auch Tag-Fields änlich wie Felder zu definieren:

```
{ datum          Field letzteZahlung  ( Tag-Field  6 Bytes )
  integer [ 12 ]_Field rate           ( Tag-Field  12 * 2 Bytes )
} ratenkauf

ratenkauf Var vax        ( teure Ratenkauf Variable 30 Bytes )
```

Entsprechend dem Vorgehen bei Feldern ist es auch hier möglich auf die einzelnen Teile ("raten") mit Hilfe eines Index (0 bis 11) zuzugreifen:

```
vax 6 rate @ .
```

Man beachte das die öffnende eckige Klammer ('[') bei der Definition von Feldern eigentlich nicht notwendig ist, sie schadet aber bei den meisten Systemen auch nicht, falls interpretiert wird.

Mit Hilfe des hier vorgestellten Konzeptes können strukturierte Datentypen elegant in Forth definiert und bearbeitet werden.
Die angegbene Implementation unterstützt jedoch weder Typüberprüfung noch Bereichsüberprüfung der Indizes zur Laufzeit. Mehrdimensionale Felder sind nur umständlich zu definieren.

Zum Abschluß noch der Code in Forth-83:

```
0  ( Datentypen )   Forth definitions decimal        ( UH 19Mar87 )
1
2    : } ( n -- ) Constant ;           0 Constant {
3
4    : Field ( offset fieldsize -- offset' )
5         Create  over , +   Does> ( offset -- offset' ) @ + ;
6
7    : Var ( size -- )  Create allot ;
8
9    : ]_Field ( offset fieldsize # -- offset' )
A       Create >r  2dup swap , , r>  * +
B       Does> ( offset index -- offset' )   2@ rot rot * +  + ;
C
D    : ]_Var ( size # -- )
E      Create over ,  * allot
F      Does> ( index -- addr )  swap over @  * + 2+ ;
```

Anmerkung zu den strukturierten Datentypen von Ulrich Hoffmann.

Diese Art und Weise, strukturierte Datentypen darzustellen, ist eine Grundübung im Forth. Interessant ist die forthtypische Syntax des Problems, die hier gefunden wurde: Benannte Felder zu erzeugen, ist in Forth eben leicht. Aber ein 'Pascal-Record' oder 'C-Struct' sind diese Datentypen eben doch nicht. Es fehlt die Datentyp-Erkennung. Ein Programm kann nicht mehr feststellen, ob ein DATUM oder eine PERSON oder ein STRING da im Feld steht. Aufzeichnen und erkennen auch der Datentypen wäre eine nützliche Erweiterung. Wer möchte sich daran versuchen?

Nun zur 'öffnende eckige Klammer': Es stimmt, solange interpretiert wird, schadet sie nicht; aber sie ist überflüssig. Hier wird eine unnötige Syntax eingeübt! Und dann passiert es - man schreibt es auch so beim Compilieren und versteht nicht mehr, wieso Forth hartnäckig mit Fehlermeldungen kommt wie HEAH? (im volksforth83) oder DEFINITION NOT FINISHED im fig-Forth. Die Namen FIELD VAR]_FIELD]_VAR sind daher nicht überzeugend. Hieran müßte noch gefeilt werden. Trotzdem, eine Basistechnik wird hier gut dargestellt.

* * *

Zur Geschichte der Forth Prozessoren,
Forth Maschinen und des Forth Chips.

Aus: TREE, Juni 1986, Autor unbekannt (?)

Schon seit einiger Zeit gibt es Bestrebungen, das Softwarekonzept FORTH in Hardware zu realisieren. In ueblichen Forthsystemen, die mit einem gewoehnlichen Microprozessor arbeiten, simuliert dieser Microprocessor eine virtuelle Forthmaschine. Der Kern dieser Simulation ist eine kurze Routine, die meist NEXT heisst. Da die virtuelle Forthmaschine recht einfach aufgebaut ist, laesst sich die Simulation auch effektiv gestalten. Das haengt allerdings ganz wesentlich davon ab, wie sehr der benutzte Microprozessor in seinem Aufbau, d.h. z.B. seiner Registerstruktur, der zu simulierenden Forthmaschine aehnelt. Bei der CPU 6809 ist die Routine NEXT mit nur zwei Befehlen (LDX ,Y++ / JMP Ä,XÜ) extrem kurz, waehrend bei den weit verbreiteten CPUs Z80 und 6502 ein vielfaches an Code noetig ist.

Warum jedoch die Forthmaschine auf einem mehr oder weniger brauchbaren Microprocessor simulieren, wenn doch ihr Aufbau einfacher ist, als der Aufbau vieler Microprocessoren?

Gewuenscht wird also eine CPU, die Forthbefehle in Hardware oder zumindest Microcode realisiert hat. Der gesammte Overhead der Simulation entfiele dann. Forth würde um ein Vielfaches schneller. Solche Forthchips entstehen gerade in den USA.

FORTH MAGAZIN:
Zu nennen wären zur Zeit folgende Produkte bzw. Projekte:
METAFORTH-BOARD, NOVIX-CHIP, JOHN-HOPKINS-CHIP, (NEUER-CHUCK-CHIP?).

COMPUTER COWBOYS
410 STAR HILL ROAD, WOODSIDE, CA 94062 (415) 851-4362

FORTHkit

The FORTHkit is a set of parts, instructions and software sufficient to build a 4MHz FORTH computer. It is provided as a kit to teach the builder to construct simple interfaces to the real world. The resulting computer provides exceptional speed (4Mips) and versatility (22 I/O lines) with minimum chips (11) and power (1 Watt).

NOVIX NC4000

The kit is designed around the Novix 120-pin FORTHchip - a 16-bit CMOS microprocessor that can execute 8 Mips at 8MHz. It boasts 1 cycle call+return timing to encourage modularity. Direct execution of Forth primitives permits simple, efficient, high-level programming.

BOARD

The unique component of the FORTHkit is a 4"x6" printed-circuit board - a 2-sided board with plated-thru holes for press-fit sockets. 17 sockets on both sides of the board support 9 memory chips (72K bytes) and 3 I/O chips (24 bits). The 4 address spaces of the FORTHchip are brought to pin&socket connectors at the board edges for further memory and I/O expansion. Boards may be plugged together into custom multi-processor architectures.

As a mother-board, the FORTHkit board has 4 I/O slots, 2 memory slots and 1 slot for each stack. Application notes describe on-board serial, printer, digital, floppy and video interfaces. Daughter-boards can provide keyboard, prom-burning, audio, video A/D, hard-disk and other interfaces.

cmFORTH

This elegantly simple version of Forth is included as both 2 4Kx8 PROMs (2732A-2) and 10 pages of source. It uses less than 2K of address space, and can compile itself as well as turnkey applications. Use is unrestricted.

PRICE

At $400 ($426 from California) the FORTHkit also includes:

- 360 Augat Holtite socket pins
- Listings of typical interface programs
- Assembly instructions
- Application notes
- Leo Brodie's Introduction to NC4000
- NC4000 spec sheet.

ASSEMBLY

A qualified builder is familiar with Forth and comfortable with the idea of building a computer. Press-fit sockets eliminate most

soldering, but you will need to solder connectors and possibly jumpers; you may want to cut traces. In case of difficulty you may want a voltmeter, logic probe or oscilloscope. Actual assembly time is only a few hours. But you will need:

a serial line (RS-232) from a host computer;
an interface program on host (Forth source provided).

You will also need the following parts. If not on your shelf, they are available from numerous sources; for example, Jameco mail order (415)592-8097.

	Jameco#	Price
5-volt power supply (~250 mA)	DC512	$4.95
6 8Kx8 static RAMs (120ns)	6264P-12	4.69
4MHz crystal oscillator	CMOS 4.000	5.95
Quad Schmidt NAND	74HC132	.79
2 1/4W resistors (3.3K, 100K)		
3 6V tantalum capacitors (1uF, 22uF, 22uF)		

These parts implement the simplest (4MHz) computer. Other versions need different parts, as described in application notes.

COMPUTER COWBOYS

Computer Cowboys represents Chuck Moore. A computer maverick, it encourages simple, reliable hardware and software. It undertakes custom software, circuit boards and chips in pursuit of this goal.

The FORTHkit is Computer Cowboys' first product. It brings next generation Forth hardware and software to individuals and schools, as well as industry.

CHUCK MOORE

Chuck Moore invented Forth about 1970 while at NRAO. Its first use was telescope control, data aquisition and analysis. He helped found Forth, Inc. in 1973 and for 10 years put Forth on dozens of different computers. This produced a portfolio of hundreds of applications involving data-base management, real-time control and numerical analysis.

In 1984 he sidestepped into IC design by hard-wiring Forth as the NC4000 for Novix, Inc. Successful first silicon led him to establish Computer Cowboys in 1985, to explore the simple hardware this FORTHchip makes possible.

Stack Akrobatik

Multiplikation mit Zwei

Bearbeitet von M. Kalus

In Bild-1 ist die Berechnung der Adresse aus einem Index und einer Startadresse dargestellt. Nehmen wir an, es sollen 2 Byte breite Daten in einem Array abgelegt werden. Die Startadresse und der Index liegen auf dem Stack. Da alle Daten 2 Bytes lang sind, liegt die gesuchte Speicherzelle INDEX 2 * Bytes von der Startadresse entfernt. Die Folge START INDEX 2 * + liefert uns die gesuchte Adresse. Es sind hier übrigens durchweg Operationen von Integer-Zahlen gemeint.

Nun ist die Multiplikation (Software und Highlevel) eine recht langsame Routine und gerade für den häufigen Fall der Multiplikation mit 2 unnötig kompliziert. Laufzeitvorteile zeigen die Alternativen in Bild-1. Dabei ist das Wort 2* Code, der die binäre Multiplikation mit 2 ausführt - shiften um ein Bit. Es ist die schnellste Art und sollte implementiert werden, wenn eine Applikation viel Datenzugriffe dieser Art enthält. Die Laufzeitunterschiede zeigt Bild-2.

```
( start index -- adresse )   2 * +       ( möglich   )
                             DUP + +     ( besser    )
                             2* +        ( am besten )

( index start -- adresse )   SWAP 2 * +  ( möglich   )
                             OVER + +    ( besser    )
                             SWAP 2* +   ( am besten )

( Bild-1)
```

```
( Verwendetes Forth System: F83, AppleII CP/M 8080)

defer TEST

: RUN  ( n -- ) 10000 0 do 1234 5678 test drop loop beep ;

: 2*+  ( n1 n2 -- n3 ) 2 * + ;
: DUP++  ( n1 n2 -- n3 ) Dup + + ;
: OVER++  ( n2 n1 -- n3 ) over + + ;
: SWAP2*+  ( n2 n1 -- n3 ) swap 2 * + ;
: CODE-2*+  ( n1 n2 -- n3 ) 2* + ;

' 2*+       is TEST   RUN  ( läuft: 11   sec )
' DUP++     is TEST   RUN  ( läuft:  5,5 sec )
' OVER++    is TEST   RUN  ( läuft:  5,5 sec )
' SWAP2*+   is TEST   RUN  ( läuft: 12   sec )
' CODE-2*+  is TEST   RUN  ( läuft:  3,5 sec )

(Bild-2)
```

A FORTH dictionary

If you use some language, even your own, you probably have at least one dictionary. Now dictionaries exist in many varieties, in Forth-connection the source screens can be considered the ultimate dictionary, but you also have shadow screens, the Handbuch and maybe more. Nevertheless I have felt the need for an always accessible source of the most important information.

A file, that always can be on the current drive, or still better in RAM, and with the sought-for information easily accessible, could be a solution; it could also be changed as the need arises.

The result is a file EXPLAIN.SCR with about 50 screens and a few Forth words to help the search. The words are ordered much as in the Handbuch and with reference to the page in this where they are described. The information given is as short as possible because the intention is to give a first and fast help for the most used words.

To give an impression of the file the index and some screens are listed below and the use illustrated by examples.

EXPLAIN context presents screen 17 with a short description of the word and a reference to part 3, page 25 of the Handbuch.

EXPLAIN /mod displays first screen 4 which informs about the stack-conditions for use of /mod, */mod and u/mod. Use of any key but Esc displays screen 5 with um/mod and m/mod. On the same screen is seen, that double precision words are only enumerated - to show which words are available, e.g. D< exists but not D>. Key Esc will stop further search.

To make possible a search of a group of words, headings are also made search-words, i.e. written in capital letters.

EXPLAIN control thus first displays screen 20 and next 21.

For a very kind and valuable advice I have to thank Bernd Pennemann.

Finn Berlev
Lillevangsvej 92
DK-3520 Farum

PS:

In your editorial in VD86 was a remark about " .. die böse Welt, die von Forth nichts wissen will..". This made me think, that a handy guide maybe could be of some help, especially to newcomers. So I have tried to make a file that could be used as a sort of dictionary in Forth essentials - with kind and critical remarks of Bernd Pennemann.

Enclosed is an article describing this file (2 copies) and, for your judgement, the file itself.

Should you find this relevant for the Forth Magazin, then I am willing to send a copy to interested readers, if I receive an addressed cover, international stamps and a formatted disk - but I only have a single-sided, 360 kB drive (SF 354).

Med venlig hilsen Farum 28/5 87

Finn Berlev

```
        Scr  0
 0                  A FORTH dictionary                FB 26/5 87
 1 A short description of some Forth words and a few related words.
 2
 3    To use the file : Load screen 1 and write   EXPLAIN <word>
 4 then the first screen describing <word> is listed.
 5    The search continues in next screen by using any key but Esc,
 6 which stops the search.
 7    All words to be searched are written in capital letters with
 8 2 trailing blanks. Use of trailing blanks speed up search of
 9 short words like I or CR, e.g. EXPLAIN I blank blank.
10    Use of ramdisk is strongly recommended because of increased
11 speed. To bring the whole file in RAM, write e.g. EXPLAIN zzz.
12    Screens 2 and 3 describe the notation used, to read them,
13 write  EXPLAIN notation.
14
15
```

```
        Scr  4
 0 SINGLE  precision                                    III-8
 1 +       -       *     (w-numbers)
 2 /      */      2/     (n-numbers)
 3   MOD  ( n1 n2 -- nr )            /MOD  ( n1 n2 -- nr nq )
 4 */MOD  ( n1 n2 n3 -- nr nq )      U/MOD  ( u1 u2 -- ur uq )
 5 1+      2+      3+      4+      2*          (w-numbers)
 6 1-      2-              4-                  (w-numbers)
 7 0       1       2       3       4      -1   (constants)
 8 NEGATE  ABS     MIN     MAX     (n-numbers)
 9                 UMIN    UMAX    (u-numbers)
10 EVEN   ( n -- n1 )   n1 = n if n even else n1 = n + 1
11
12 0<      0=      0>      0<>                         III-6
13  <       =       >
14 U<              U>
15 UWITHIN  ( u u1 u2 -- flag )        true if u1 <= u < u2
```

```
        Scr  5
 0 DOUBLE  precision                                    III-10
 1 D+     D-     D*     DNEGATE     DABS
 2 D0=    D=     D<
 3
 4 MIXED precision
 5 EXTEND  ( n -- d )
 6 UM*     ( u1 u2 -- ud )          M*     ( n1 n2 -- d )
 7 UM/MOD  ( ud u -- ur uq )        M/MOD  ( d n -- nr nq )
 8 UD/MOD  ( ud u -- ur udq )
 9
10 LOGIC                                               III-6
11 AND     OR      XOR      NOT
12 TRUE  = -1                      FALSE  = 0
13 ON   ( adr -- ) (adr) true      OFF  ( adr -- ) (adr) false
14
15
```

```
        Scr 17
 0 VOCABULARY                                           III-25
 1 WORDS     list words in actual vocabulary
 2 ORDER     list context vocabularies (used in dictionary search)
             and  current vocabulary (where new words go)
 3 CONTEXT   ( -- adr )  of first vocabulary to be searched
 4 CURRENT   variable, pointer to current vocabulary
 5 DEFINITIONS   make context vocabulary also current
 6 VP        ( -- adr ) array with number of and addresses of
                        context vocabularies
 7 VOC-LINK  U, startadr for a linked list of all vocabularies
 8
 9 ALSO  <voc>  <voc> is added to context vocabulary
10 TOSS         the last vocabulary is removed from context
11 ONLYFORTH    set ORDER:  FORTH FORTH ONLY  FORTH
12 SEAL         remove ONLY from context; for making applications
13
14
15
```

```
        Scr 20
 0 CONTROL  1                                           III-29
 1 BEGIN  ... flag  WHILE  ... REPEAT                      I C
 2 BEGIN  ... flag  WHILE  ... flag  UNTIL                 I C
 3 Zero or more whiles
 4
 5 flag  IF  ... ELSE  ... THEN       else optional        I C
 6
 7 limit start  DO  ... LOOP                               I C
 8 limit start  DO  ... n  +LOOP                           I C
 9 ?DO   can replace  DO, then no loop if limit = start    I C
10 BOUNDS  ( start count -- limit start )
11 I   ( -- w )  copy of loop index                          C
12 J   ( -- w )  copy of outer loop index in nested loops :  C
13               do ... do .. J.. loop ... loop
14 LEAVE         leave loop immediately ( only in loops )    C
15
```

```
        Scr 35
 0 I/O  devices:                                        III-44
 1 0  Prt,  parallel port (printer)
 2 1  Aux,  RS232 serial port (modem)
 3 2  Con,  keyboard and monitor
 4 3  Midi,
 5 4  Ikbd, keyboard
 6
 7 BCONIN    ( dev# -- char )  wait till a char is ready in      A
 8                             input-device dev# . 4 is not allowed
 9 BCONOUT   ( char dev# -- ) send char to output-device dev#    A
10 BCONSTAT  ( dev# -- flag ) true if dev# is ready to send a   A
11                              character 0 and 4 is not allowed
12 BCOSTAT   ( dev# -- flag ) true if dev# is ready to receive  A
13
14 STANDARDI/O  make I/O devices as they were at last SAVE   III-54
15
```

Einige Sreens aus EXPLAIN.SCR als Beispiel. Das ganze File kann beim Autor auf Diskette bezogen werden.

Listing des Files EXPLAIN.SCR

```
 0                A FORTH dictionary              FB 26/5 87
 1 \ Explain a word
 2 NOTATION  1                                        III-1
 3 NOTATION  2
 4 SINGLE  precision                                  III-3
 5 DOUBLE  precision                                  III-10
 6 STACK  and RETURNSTACK                  III-12     III-14
 7 MEMORY  and STRINGS  1                             III-8
 8 STRINGS  2
 9 STRINGS  3
10 STRINGS  4
11 NUMBERS
12 STRING  to NUMBER                                 III-15
13 DEFINING  words                                   III-19
14 DICTIONARY  1                                     III-22
15 DICTIONARY  2                                     III-22
16 DICTIONARY  3                                     III-22
17 VOCABULARY                                        III-25
18 VOCABULARIES
19 HEAP                                              III-27
20 CONTROL  1                                        III-28
21 CONTROL  2
22 COMPILER  1                                       III-31
23 COMPILER  2
24 INTERPRETER  1                                    III-33
25 INTERPRETER  2                                    III-33
26 INTERPRETER  3                                                III-33
27 ERROR                                                         III-36
28 MASS STORAGE  1                                               III-40
29 MASS STORAGE  2                                               III-40
30 MASS STORAGE  3
31 MASS STORAGE  4
32 USER  variables
33 MULTITASKING  1  include TASKER.SCR                           III-48
34 MULTITASKING  2  TASKER.SCR                                   III-48
35 I/O  devices!                                                 III-44
36 INPUT  1                                                      III-51
37 INPUT  2                                                      III-44
38 INPUT  3                                                      III-51
39 OUTPUT  1                                                     III-51
40 OUTPUT  2                                                     III-44
41 OUTPUT  3                                                     III-51
42 LONG  adressing                                               III-56
43 PRINTING  words in FORTH vocabulary!                      printer.scr
44 PRINTER  vocabulary!                                      printer.scr
45 DEBUG  TRACE                                                   II-41
46 GRAPHICS  vocabulary  1                                   line_a.scr
47 GRAPHICS  vocabulary  2                                   line_a.scr
48 GRAPHICS  vocabulary  3                                   line_a.scr
49 EDITOR
50 ASSEMBLER                                                       A-17
51 DISASSEMBLER  include DISASS.SCR                                A-31
52 MISCELLANOUS  1                                               III-38
53 MISCELLANOUS  2
```

Explain a word

```
\needs ramdisk.scr cr .( works much better with RAMDISK.SCR !)

: (explain  ( $ n -- offset f ) \ search $ in block n
   swap count rot block b/blk search ;

: show  ( n -- ) \ present screen with word
   page list c/l /mod 1+ swap 3+ at ;

: explain  ( name ( -- ) \ takes name from inputstream
   isfile push fromfile push scr push caps push
   explain.scr caps off 0 word capitalize capacity 2
   do dup i (explain
      if i show key $11 0 at $ff and #esc = if leave then
      else drop then loop drop ;
```

(Geschrieben fuer und mit volksFORTH-83.)

Aus: FORTH DIMENSIONS, Volume VIII, No.6, January/February 1987

State of the Standard

Marlin Ouverson
La Honda, California

Forth standards have arisen, throughout the history of the language, from self-governing committees comprised of expert users of Forth. Participation was open, and becoming a voting member was a matter of meeting minimal requirements. Coming from different backgrounds, these experts often had deeply vested opinions about what should and should not be part of a common Forth kernel, about how those functions would operate and what their names would be, and about standardization itself. The Forth Standards Team (FST) has been the arena in which these elements converge and, at times, diverge. The team has had a benign relationship with the Forth Interest Group, but operates independently. And while it has been the subject of harsh criticism, it has also received a great deal of praise.

What have been the rough spots? One easy target is the malleability that permits Forth to become what each programmer or developer needs (or wants) it to be. More generality and less bulk seem to be called for in a Forth standard than in languages frozen at the moment of creation: Forth systems grow with their users, and those users may resent being told that the programs they develop are non-standard. And creating *new* Forth standards brings the added wrath of both vendors and users if it creates incompatibilities with previously standard systems.

Most of the history of Forth standards has been recorded in these pages and elsewhere, in articles and letters to the editor. We will not attempt a historical summation, but present a sampling of the ideas in active circulation at this time. The amount of material precludes reproduction *in toto*; what follows is a general survey, quoting liberally from documents in our files. To get a reading on the opinions of one cross-section of the Forth community, see "FORML '86 in Review," elsewhere in this issue. We must also acknowledge that, for many people working independently or on some in-house systems, the issue of standards may be only of secondary importance. If, however, this topic is of concern to you, the best way to be fully informed and to participate is to make contact with the people and organizations directly involved.

ANSI Standard Requested

Elizabeth Rather, President of FORTH, Inc., wrote in December to say that a project proposal for an ANS Forth had been filed with ANSI. The group that filed the proposal consisted of Ms. Rather (also an FST member); Don Colburn (FST member, Creative Solutions, Inc.); W.B. Dress (Oak Ridge National Laboratory); Ray Duncan (Laboratory Microsystems, Inc.); Burt Feliss (IBM Corporation); Charles Moore (inventor of Forth, Computer Cowboys); Dean Sanderson (FST Referee, FORTH, Inc.); Gerald Shifrin (MCI Telecommunications Corp.); and Martin Tracy (FIG board member, FORTH, Inc.).

Ms. Rather wrote, "To date we have not heard from ANSI. If and when they do form a Technical Committee for Forth, it will be publicly announced according to their standard procedures, and everyone who is interested and willing to make the required commitment will be able to participate.

"According to ANSI rules, a voting member of a technical committee pays a fee of $175 to ANSI and must attend at least two out of three meetings to retain voting status. The first meeting is usually held at ANSI headquarters in Washington, D.C. Subsequent meetings are held in various parts of the country. Meetings typically occur four times a year for four or five days each. The C committee has been working for five years."

First of all, this means that ANSI has to accept the proposal. And the proposal group does not intend revolution, for the formal proposal states, as the first item in the program of work, "Identify and evaluate common existing practices in the area of the Forth programming language." Under the category of implementation impacts, the proposal points out current incompatibilities among popular Forth dialects and says, "While the Forth-83 Standard has stabilized the language to a great extent, it has proven too restrictive and machine-dependent. Assuming the ANS Forth standard confines itself to such changes as are necessary to resolve the problems in Forth-83, the effect on current practice will be modest." It also projects a five-year useful life of such an ANS standard.

It has long been the view of some that an ANS Forth standard would greatly boost the language's acceptance in the corporate and government world. Others argue that a Forth system stands on its own merits, and that going to ANSI would remove the standardization process too far from the Forth community. The project proposal cited above states, "Preserving machine independence and maintaining a close liaison with any other Forth standardization efforts should prevent problems related to restraint of trade and public interest." It concludes, "If any Forth standard committees are formed by the ISO or IEEE, a close liaison should be formed."

IEEE Action Requested

George Shaw of Shaw Laboratories, Ltd., points out that there is more than one route to an ANSI standard. In one letter, he said, "It took the thirty or so individuals directly involved (and probably several times as many lobbyist and mail participants) in Forth-83 to represent the diversity of implementations and usage. Some important considerations may only have been represented by a single individual.... Considering a standards group with such a small number of participants would end up standardizing a particular group of vendor's implementations at the expense of others. The CBEMA effort, I fear, will produce such a small group."

Shaw explained that ANSI itself doesn't create standards, but endorses them. It is primarily concerned with whether a standards document was obtained from one of their usual channels, like CBEMA (the route chosen by

Rather, et al.) and IEEE. The actual procedure for creating a standard document is dictated by the particular channel. Shaw feels that IEEE can provide for participation and meaningful input from a broader cross-section of the Forth community. Under its own rules, participation by mail must be allowed, and individuals participating by mail are allowed to vote. Teleconferencing and other options are available in consideration of the difficulties of individual participation. CBEMA requires four to six meetings per year, of four or five days each; and the voting membership is virtually restricted to organizations and businesses, not individuals. Shaw's letter continues, "Meeting [CBEMA] requirements to maintain voting privileges would cost approximately $3000 a year in dues and travel expenses, not to mention lost wages or use of vacation time...."

Shaw feels that a CBEMA effort to developing an ANS Forth standard is unnecessarily restrictive, considering how widely expertise is distributed throughout the Forth community. He said that to CBEMA, in a letter asking them not to approve the proposal they received. He points out that the group who wrote the proposal is made up mostly, if not entirely, of current or former employees, customers or subcontractors of FORTH, Inc. He wrote, "They may be well intentioned, but I do not believe this group represents the interests of the Forth community and vendors at large."

This matter was presented at a January meeting of the Microcomputer Standards Committee of the IEEE. Shaw says, while the members of that committee "wished to avoid the possibility of and the political problems involved in having a joint IEEE/CBEMA committee ... the group voted with no dissension (nineteen yea, four abstentions) to untable and replace a motion made in 1981 for a PAR (program action request) and to additionally request that the members who are also voting members in CBEMA vote against approval of the ANS Forth project." In the IEEE, a PAR is the first step in getting a standard project going.

Forth Standards Team

The above actions may have been prompted by dissatisfaction with the Forth-83 Standard itself, with the process used by the Forth Standards Team or with the continuing unrest in some parts of the Forth community over standardization in general. Vocal dissent over the latest standard seems to have found a home on the East Coast Forth Board (703-442-8695, up to 2400 baud). Sysop Gerald Shifrin sent me standards discussions archived on diskettes that, when printed, amounted to a stack of paper larger than most book manuscripts. I sent a copy of the diskettes to Guy Kelly, FST chairman, to get his reactions.

According to Kelly's analysis, much of the debate over Forth standards on the East Coast Forth Board has been from a vocal few (two participants together account for nearly half the 780 messages; the overall average is twenty-eight messages per participant). Most of the messages fall into a few categories, the first of which is complaints about Forth-83. On the technical side, dissension focuses primarily on floored division, **DO LOOP**s, **FIND**, alleged ambiguities and, in particular, new or modified actions assigned to word names already used. About the last item Kelly says, "Giving old names new meanings was considered the most offensive action that was taken. I agree that it was a radical step and one which should never be repeated! However, it was not done in ignorance, but only after a great deal of careful consideration.

"Something that seems to be completely overlooked in the current discussions is that all the attempts to produce a standard prior to Forth-79 were preliminary gropings and that Forth-79 was fatally flawed....

"Now if the major vendors had said no to the 'obnoxious' changes between the 1979 and the 1983 standards, the standards team probably would have produced a somewhat different Forth-83 Standard (we had received twenty-two yes votes and zero no votes from the twenty-six voting members when the standard was finally released)."

Other topics of discussion from the electronic, standards debate include organization of the FST, suggestions for future standardization efforts and specific work needed (such as surpassing sixteen bits, local or stack variables, data and programming structures, bit manipulation, vectored I/O, quans and transient headers). Extensions have been requested to provide floating-point math, operating system interfaces, files, graphics, strings and math/statistics packages.

Kelly observes that most disliked by the electronic conferees are: floored division, dumb tick ('), smart tick, new **LEAVE**, Forth-83, Forth-79, the small wordset and new actions associated with old word names. He contrasts those with the things conferees have said they like: floored division, dumb tick, smart tick, new **LEAVE**, Forth-83, Forth-79 and the small wordset.

Regarding FORTH, Inc.'s part in the history of Forth standardization, Kelly says, "FORTH, Inc. was forcefully involved in the standards efforts. They hosted the 1977 and 1978 meetings and had three or four participants ... at every standards meeting. FORTH, Inc. has tended to track the various standards, and because Forth is still evolving, the standards have also tended to track the work at FORTH, Inc. The following quote from a May 1978 FORTH, Inc. bulletin entitled *'FORTH-77' Implementation on FORTH, Inc. Systems* may be of interest:

"'We feel that the adoption of standards is an extremely important step in the growing acceptance of Forth, so long as these represent a "minimum vocabulary" with options rather than being interpreted in a restrictive sense.'

Kelly continues, "The fact that Mr. Moore does not personally feel bound by a standard and is continually evolving his own version of the language he invented is, I believe, all to the good."

Finally, the FST chairman calls our attention to these words from the forward to the *Forth-83 Standard*:

"Forth's extensibility allows the language to be expanded and adapted to special needs and different hardware systems. A programmer or vendor may choose to strictly adhere with the standard, but the choice to deviate is acknowledged as beneficial and sometimes necessary. If the standard does not explicitly specify a requirement or restriction, a system or application may utilize any choice without sacrificing compliance to the standard provided that the system or application remains transportable and obeys the other requirements of the standard."

✶

Ultimate CASE Statement

Wil Baden
Costa Mesa, California

Many citizens of the Forth community have lamented the lack of a **CASE** statement in standard Forth language specifications. Since the first rule of Forth programming is, "If you don't like it, change it," there have been many proposals, and *Forth Dimensions* even held The Great CASE Contest in Volume II. Although the winning entry of that contest, submitted by Charles Eaker, has been widely implemented and even offered as part of many vendors' systems, the flood of proposals has not ceased. There have been many articles and letters on the subject in *Forth Dimensions.*

All proposals to date have had problems. Portability is one. Another is that they all have been too specialized and restricted in their area of application. Generalization is accomplished by designing another special case of **CASE.**

Strictly speaking, a **CASE** statement is unnecessary. It is "syntactic sugar" to make a program easier to write, read and understand. It is so helpful in doing this that it is a standard feature of all other modern programming languages.

Figure One-a is a rather futile program written in C to illustrate a common pattern of logical decisions in many programs. ("= =" is "equal to" for comparing two things, to distinguish it from "=" for assignment as in Fortran or Basic.) An equivalent Forth version would look something like Figure One-b.

Most people will agree that Figure One-a would be better written as in Figure Two-a. An even better way is found in some dialects of C, illustrated by Figure Two-b. In this extension, following syntax from Pascal, values separated by "," indicate a set of values, and values separated by ".." indicate a range.

Some Forth proposals have one definition for individual values and another definition for a range of values. There would have to be another definition for a set of values. No earlier Forth proposal that I know of allows sets and ranges together, as in:

case 2..3, 12:

What is proposed here is a single **CASE** statement for Forth which will include all these variations, and many more, that can be implemented in fig-FORTH, Forth-79, Forth-83 and any other Forth.

Figure Two-a would look as shown in Figure Three. Let's add two more spoons of syntactic sugar, as in Figure Four. As has been noted elsewhere, too much syntactic sugar causes semantic diabetes. Our **CASE** is sweet enough. Figure Five is an example to show some of the possibilities.

Now for a real life example. Figure Six is a recension of a word in John James' "Universal Text File Reader" (*Forth Dimensions* VII/3). One of my favorite examples is "Thirty days hath September, April, June and November ...." See Figure Seven.

If **NUMBER** in your system is vectored, you may want to replace it in some

```
craps(n)
int n;
{  if (n == 7)
      printf("You win");
   else if (n == 11)
      printf("You win");
   else if (n == 2)
      printf("You lose");
   else if (n == 3)
      printf("You lose");
   else if (n == 12)
      printf("You lose");
   else printf("%d is your point",n);
}
```

Figure One-a

```
: CRAPS ( n -- )
   DUP 7 =
   IF DROP ." You win"
   ELSE DUP 11 =
      IF DROP ." You win"
      ELSE DUP 2 =
         IF DROP ." You lose"
         ELSE DUP 3 =
            IF DROP ." You lose"
            ELSE DUP 12 =
               IF DROP ." You win"
               ELSE . ." is your point" THEN
   THEN THEN THEN THEN THEN ;
```

Figure One-b

```
craps(n)
int (n);
{  switch(n) {
   case  7:  printf("You win"); break;
   case 11:  printf("You win"); break;
   case  2:  printf("You lose"); break;
   case  3:  printf("You lose"); break;
   case 12:  printf("You lose"); break;
   default:  printf("%d is your point",n);
   }
}
```

Figure Two-a

```
craps(n)
int n;
{  switch(n) {
   case 7, 11:     printf("You win"); break;
   case 2..3, 12: printf("You lose"); break;
   default:        printf("%d is your point",n);
   }
}
```

Figure Two-b

```
: CASE   DUP ;

: CRAPS  ( n -- )
   CASE   7 = IF DROP ." You win" EXIT THEN
   CASE  11 = IF DROP ." You win" EXIT THEN
   CASE   2 = IF DROP ." You lose" EXIT THEN
   CASE   3 = IF DROP ." You lose" EXIT THEN
   CASE  12 = IF DROP ." You lose" EXIT THEN
   . ." is your point" ;
```

Figure Three

```
: OF   ( n flag -- ) [COMPILE] IF   COMPILE DROP ; IMMEDIATE
: =OR  ( n flag n -- n flag ) 2 PICK = OR ;

: CRAPS  ( n -- )
   CASE  7 =  11 =OR          OF ." You win" EXIT THEN
   CASE  2 3 BETWEEN  12 =OR  OF ." You lose" EXIT THEN
   . ." is your point" ;
```

Figure Four

```
: WHATEVER  ( n --)
   CASE  0=                          OF ." Zero" EXIT THEN
   CASE  0<                          OF ." Negative" EXIT THEN
   CASE  DUP 1- AND 0=               OF ." Power of 2" EXIT THEN
   CASE  ASCII 0  ASCII 9  BETWEEN OF ." Digit" EXIT THEN
   CASE  ASCII ,  ASCII /  BETWEEN
         ASCII : =OR          OF ." Punctuation ,-./: " EXIT THEN
   DROP  ." Whatever" ;
```

Figure Five

```
: ?OUT  ( c -- )  127 AND
   CASE  0=  13 ( return) =OR
     OF  ?NEW-LINE  EXIT THEN
   CASE  10 ( linefeed) =  12 ( formfeed) =OR
     OF  #BLANK-LINES @ 0=
         IF  ?NEW-LINE  THEN
         EXIT THEN
   0 #BLANK-LINES !
   CASE  32 <
     OF  ( Do nothing.)  EXIT THEN
   EMIT ;
```

Figure Six

Der CASE-Wettbewerb 1980/81 in 'Forth Dimensions',Vol.II

Es erschienen folgende Artikel dazu (Volume/Seite):

:CASE II/41
CASE AND PROD CONSTRUCTS II/53
CASE AS DEFINING WORD III/189
CASE AUGMENTED III/187
CASE CONTEST STATEMENT II/73
CASE IMPLEMENTATION II/60
CASE STATEMENT II/55, II/81, II/82,II/84,II/87
CASE, SEL AND COND STRUCTURES II/116
CASES CONTINUED II/187

Und es wurden diese Varianten definiert:

:CASE II,41
CASE, a Generalized Structure III/190
CASE by Bochert/Lion II/50
CASE by Brecher II/53
CASE by Brothers II/55
CASE by Eaker II/37
CASE by Emery II/60
CASE by Fittery II/62
CASE by Kattenberg II/67
CASE by Lyons II/73
CASE by Munson II/41
CASE by Perry II/78
CASE by Powell II/81
CASE by Selzer II/82
CASE by Wilson II/85
CASE by Witt&Busler II/87
DO-CASE by Elvey II/57
DO-CASE by Giles II/64

Praktisch durchgesetzt hatte sich die Version von Charles Eaker.
Die Reihe wurde dann beendet. Doch immer wieder tauchen neue, meißt recht spezielle CASE auf. Wil Baden diskutiert hier das CASE erneut unter dem Gesichtspunkt: Überflüssig, aber man kann manche Programme damit leichter schreiben und vor allem auch besser lesen.

```
: LEAPYEAR?  ( -- tf : true when the year is a leap year.)
 #YEAR @
   CASE 400 MOD 0= OF  TRUE  EXIT THEN
   CASE 100 MOD 0= OF  FALSE EXIT THEN
   CASE   4 MOD 0= OF  TRUE  EXIT THEN
   DROP  FALSE ;

: DAYS  ( month# -- days-in-month )
   CASE  9 =  4 =OR  6 =OR  11 =OR  OF  30  EXIT THEN
   CASE  2 = NOT                    OF  31  EXIT THEN
   DROP  LEAPYEAR? IF  29  ELSE  28  THEN ;
```

Figure Seven

```
: CBASE!  ( a c -- a' )
   CASE  ASCII $ =  OF  HEX     1+ EXIT THEN
   CASE  ASCII @ =  OF  OCTAL   1+ EXIT THEN
   CASE  ASCII % =  OF  BINARY  1+ EXIT THEN
   CASE  ASCII & =  OF  DECIMAL 1+ EXIT THEN
   DROP ;

: BASE-NUMBER  ( a -- d )
   BASE @ >R   DUP 1+ C@ CBASE!
   NUMBER?   R> BASE !   0= ABORT" ?" ;
```

Figure Eight

```
HEX
: CLASSIFY  ( n -- )
   CASE  20 <  7F =OR       OF ." Control character" EXIT THEN
   CASE  20 2F BETWEEN
   OVER  3A 40 BETWEEN OR
   OVER  5B 60 BETWEEN OR
   OVER  7B 7E BETWEEN OR  OF ." Punctuation"        EXIT THEN
   CASE  30 39 BETWEEN     OF ." Digit"              EXIT THEN
   CASE  41 5A BETWEEN     OF ." Upper case letter"  EXIT THEN
   CASE  61 7A BETWEEN     OF ." Lower case letter"  EXIT THEN
   DROP                       ." Not a character" ;
```

Figure Nine

```
CREATE CASE   ' DUP ( CFA ) @ ' CASE ( CFA ) !
```

Figure Ten-a

```
: =OR   ( n tf n -- n tf ) 3 PICK = OR ;
```

Figure Ten-b

```
: =OR  ( n tf n -- n tf ) >R  OVER R> = OR ;
```

Figure Ten-c

```
: WITHIN   ( n n1 n2 -- tf : true when n1 <= n & n < n2.)
   OVER - >R   -  R> U< ;
: BETWEEN  ( n n1 n2 -- tf : true when n1 <= n & n <= n2.)
   WITHIN 1+ ;

: ASCII  ( ^ c -- c : integer value of character c.)
   BL WORD COUNT 1- ABORT" ?"   C@ STATE @
   IF  [COMPILE] LITERAL  THEN ; IMMEDIATE
```

Figure Eleven-a

```
: HEX  ( -- ) 16 BASE ! ;
: OCTAL  ( -- ) 8 BASE ! ;
: BINARY  ( -- ) 2 BASE ! ;
: DECIMAL  ( -- ) 10 BASE ! ;

: NUMBER?  ( addr -- dn tf ) 0 0 ROT CONVERT C@ BL = ;
```

Figure Eleven-b

applications with a version that selects the numerical radix according to the first character. Figure Eight implements a convention used on Motorola systems (e.g., 68000). Laxen's **CLASSIFY** example (*FD* VII/1) can be written without redundant classes with no additional definitions, as in Figure Nine.

Since **DUP** is assembler code, in most systems you can optimize its definition with something like that in Figure Ten-a. The Forth-79 definition of **=OR** is given in Figure Ten-b. If you do not have **PICK**, as in fig-FORTH, or if **PICK** is not an assembler code definition, see Figure Ten-c.

A **CASE** statement in any programming language is intended for a series of tests to classify a value. To do this in other languages without using a **CASE** structure would require repeating the value at each test, giving a tedious appearance to the source. In Forth, the data stack allows us to avoid such explicit references to the value. In Forth, a **CASE** statement has the pattern **DUP** ... **IF DROP** .... We have sweetened this to **CASE** ... **OF** ....

The trivial nature of the implementation emphasizes that a **CASE** statement is not essential to Forth. Those Forth practitioners who pride themselves on how lean and mean their Forth is will find it superfluous. My intent is not to propose this definition of **CASE** for standardization; but on the other hand, any further **CASE** proposal should be as simple to implement, as portable and as powerful.

Auxiliary Definitions

You may already have some of these. Your definitions may be different from those shown in Figure Eleven-a. **#BLANK-LINES** and **?NEW-LINE** are words peculiar to the application. **#BLANK-LINES** is a variable counting the number of successive blank lines. **?NEW-LINE** does a CR when the value of **#BLANK-LINES** is less than two.

Figure Eleven-b provides definitions for several fundamental Forth words. It also presents a naive version of **NUMBER?** that ignores details such as sign and punctuation, and is not intended for actual use.

Practical Considerations for
Floating-Point Arithmetic

Richard Wilton
Marina del Rey, California

In most high-level languages, whether or not to use floating-point arithmetic is not even a question. Fortran, PL/1 or C programmers simply take for granted that when they wish to compute with real numbers, the language they are using offers the tools to do so. The presence of arithmetic data types in such high-level languages allows the selective use of integer or real arithmetic.

In contrast, Forth deals with objects on a somewhat less abstract level. A Forth programmer must always be aware of the low-level representation of real numbers and the manner in which arithmetic operators are implemented. These considerations are much less important to programmers in most high-level languages.

This article discusses some of the practical points involved in doing Forth floating-point arithmetic. It starts by covering the salient low-level features of floating-point system design in Forth. The simple source code examples which follow illustrate some of the points to consider in designing real arithmetic into a Forth application.

Real-Number Representation

One of the first questions the implementor of floating-point numbers has to solve is that of the representation of real numbers. The usual representation is a simple data structure containing an exponent (sometimes called the "characteristic"), a significand ("mantissa") and a sign bit. An example is shown in Figure One.

With an eight-bit exponent, a twenty-three-bit significand and one sign bit, this real-number representation could be stored in two sixteen-bit words on the usual Forth stack. Many similar representations can be used in Forth floating-point implementations.

A Forth systems programmer chooses the representation best suited to a particular hardware and software situation. For example, some representations are more easily used in software floating-point primitives, whereas others correspond to the representation used by a floating-point coprocessor such as the AMD 9511 or the Intel 8087, or to that used by firmware routines such as those in the Apple Macintosh or in the IBM PC's BASIC ROM.

A Forth application programmer who uses floating-point arithmetic must be aware of the representation used, because the dynamic range and accuracy of real numbers is implicit in their representation. Also, if you wish to manipulate real numbers with standard Forth operators such as **2@** or **CMOVE**, you must know how many bytes of storage are required for each real number.

Manipulating Real Numbers

Another important point to consider when you use floating-point arithmetic in Forth is the problem of where to place real numbers so that they can be manipulated conveniently. Because integer arithmetic is sufficient for Forth's memory-conserving, threaded code interpreter, the Forth virtual machine is implicitly biased towards performing integer arithmetic. Integrating real numbers and floating-point operators into the standard Forth system thus demands careful consideration.

There are two common solutions to this problem. One is to maintain real numbers on Forth's parameter stack. The other is to design a separate real-number stack which is tightly integrated into the standard Forth interpretive system. Both approaches are viable.

Using the Parameter Stack. For most purposes, there is no reason to avoid placing real numbers on the parameter stack, even though they are almost certainly represented as thirty-two-bit, forty-eight-bit or even sixty-four-bit numbers. After all, the usual Forth stack is already cluttered with data items of various sizes and types, including eight-bit characters, sixteen-bit signed and unsigned integers, thirty-two-bit integers and addresses of various sizes.

An advantage to manipulating floating-point data on the Forth parameter stack is that the usual stack and memory operators can be easily adapted to handling real numbers. For instance, if a real number is represented in sixty-four bits, then

```
: FDROP ( r -- )
DROP DROP DROP DROP ;
```

is exactly analogous to **DROP** for sixteen-bit integers or to **2DROP** for thirty-two-bit integers. Similar operators, such as **FDUP**, **FSWAP**, **FPICK** and so on can be defined in terms of the standard Forth stack words.

A common problem is that the parameter stack can quickly become crowded, particularly when sixteen-bit integers and addresses must be maintained on the stack at the same time as real numbers. Bugs introduced by inaccurate stack operations (for example, **SWAP** instead of **FSWAP**) can be notoriously difficult to track down.

Using a Separate Stack. In an effort to avoid stack clutter, some implementors of Forth floating-point support simply maintain all real numbers on a separate, dedicated stack. This design makes life much easier for programmers who make heavy use of the parameter stack.

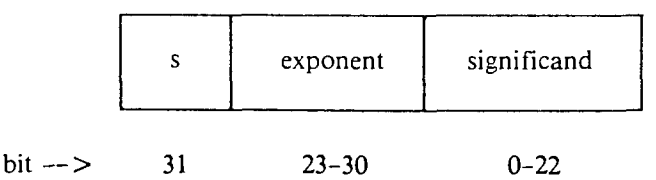


Figure One

The separate stack approach can also lead to significantly improved performance if it is supported in hardware. For example, the Intel 8087 arithmetic coprocessor maintains its own stack. (The stack is only eight deep, but this is sufficient for most applications.) A separate real-number stack thus maps directly onto the hardware, which simplifies the low-level software primitives and leads to increased execution speed in application programs.

In practice, neither approach to floating-point stack design has proved to be unequivocably better. Other considerations, including source code readability, portability and the asymmetry of floating-point hardware with standard Forth system design, lead to compromises in system complexity and in execution speed.

```
( STEST -- Scaled arithmetic version )
: AREA ( radius -- area )
   DUP *                                                  \ r^2
   355 113 */ ;                                       \ pi * r^2

( USTEST -- Unsigned scaled arithmetic version )
: AREA ( radius -- area )
   DUP *                                                  \ r^2
   355 UM* 113 UM/MOD SWAP DROP ;                     \ pi * r^2

( FTEST -- Floating point version )
: AREA ( radius -- area )
   DUP M* D>F  FPI F* ;                               \ pi * r^2

( F87TEST -- version which uses 8087 stack )
: AREA ( radius -- area )
   0                \ convert to double (8087 "short integer")
   IS>AP APDUP (FMULP)                        \ r^2 on 8087 stack
   (FLDPI) (FMULP) AP>FL ;                            \ pi * r^2


( Timing loop )
: TEST ( -- )
   !TIMER
  100 0 DO
   101 1 DO  I AREA DROP  LOOP          \ substitute FDROP in ..
       LOOP                        \ .. floating point versions
   .TIMER ;
```

Table One. Source Code Examples.

STEST	USTEST	FTEST (SFP)*	FTEST (8087)	F87TEST
5.16	3.46	75.63	5.88	3.63

Table Two. Timings for 10,000 executions of AREA (IBM PC, 4.77 MHz 8088).

STEST	USTEST	FTEST (SFP)*	FTEST (8087	F87TEST
1.16	0.71	18.34	2.26	1.53

Table Three. Timings for 10,000 executions of AREA (IBM PC AT, 8 MHz 80286).

*SFP means "Software Floating Point."

Floating-Point Operators

Most programmers perform floating-point arithmetic in Forth with operators that are analogs of the standard Forth integer arithmetic operators. Floating-point operators with analogous names (e.g., **F+**, **FDUP**, **F@**) perform functions analogous to the standard integer operators. It is easy to program "intuitively" with this type of system.

Some programmers prefer to redefine the standard integer operators so that they work with real numbers instead. These redefined operators are maintained in a separate vocabulary. This approach allows a given piece of source code to be used with either number type, simply by switching vocabularies. Also, the same set of operators can be used for either integer or real arithmetic, just as they are in Fortran and other high-level languages.

The disadvantages of both approaches are clear. Using a parallel set of operators adds two or three dozen new words to a language which already demands familiarity with several hundred words. However, redefining existing Forth integer operators to handle real numbers also creates problems. A program which manipulates both data types simultaneously soon becomes littered with vocabulary changes which obscure the functional meaning of the source code.

Other Considerations

Forth systems programmers must consider many other issues of floating-point implementation, including accuracy, rounding, representation of values which cannot be exactly expressed in binary, infinity, error trapping (division by zero, invalid arguments to trigonometric functions) and so on. Such implementation details are often irrelevant to an application programmer. However, in many instances, knowledge of the exact behavior of the floating-point package is critical to debugging as well as to obtaining accurate results.

A Simple Example

At this point it is worthwhile to examine some source code. Apart from superficial differences in notation, it is important to observe the implicit differences between integer and floating-point arithmetic when each is used for computation of fractional quantities. Although there are applications which by nature demand the use of either integer or real arithmetic, situations frequently arise in which the choice is affected by stylistic or performance considerations.

The simple example in Table One calculates the area of a circle four different ways. The first two, **STEST** and **USTEST**, use scaled integer arithmetic. The value for pi is the well-known ratio 355/113, which is accurate to six decimal places. The scaling in **USTEST** looks slower but runs faster because it does not use */* and thereby avoids the overhead of floored division.

The second pair of examples, **FTEST** and **F87TEST**, use floating-point arithmetic to do the same work. **FTEST** is written with a set of floating-point operators which parallel the usual integer operators. It uses the Forth parameter stack for all real arithmetic, so integers and real numbers coexist on the stack at the same time. The last example, **F87TEST**, uses the Intel 8087's separate stack to hold real numbers for intermediate calculations.

A comparison of the source code reveals little on the surface apart from the somewhat obscure operators used to manipulate the 8087 stack directly. There is, however, a great deal of difference in dynamic range and in precision implied by the use of floating-point operators. Any increase in precision of the integer versions **STEST** and **USTEST** would require additional scaling operations with a significant performance degradation as a consequence, as well as additional code required to support scaling.

Tables Two and Three contain typical performance data. Most of the differences in timing between the examples is due to the time required for multiplication by pi. The timing loop calls the **AREA** routine 10,000 times and uses the computer's system clock (accurate to about 0.06 seconds on an IBM PC) as a timer.

The poor performance of **FTEST** when real arithmetic is carried out in software (SFP) stands out in sharp contrast to the other results. (Nevertheless, it is still a bit faster than interpreted BASIC!) What is striking is that the speed of floating-point arithmetic using a hardware coprocessor is quite close to that of integer arithmetic, yet the degree of precision and dynamic range achievable with the use of floating-point arithmetic is far beyond the capabilities of integer arithmetic, scaled or not.

Practical Experience

It would be wrong to extrapolate from these simple timing data that real arithmetic will always be just about as fast as integer arithmetic in Forth. The point is that the performance penalty for using floating-point arithmetic in Forth is negligible in situations where an application demands precision and dynamic range. There is no reason to use scaled arithmetic to avoid decreased run-time performance if the degree of performance degradation is not critical and if significantly increased source code complexity results.

This observation has been thoroughly demonstrated in real-world situations. Floating-point Forth programs have been successfully utilized in applications such as high-level display graphics, real-time engineering telemetry processing and industrial quality-control analysis. A Forth program which uses floating-point arithmetic is often the best approach to an application which demands real-number processing as well as interactive hardware control.

With inexpensive, widely available floating-point hardware, real numbers can be handled in a sophisticated manner without sacrificing either speed or the many conveniences of the standard Forth interpretive environment. Furthermore, in well-integrated systems such as the Apple Macintosh, it behooves a Forth programmer to take advantage of readily available firmware support for real arithmetic. With a critical eye to the factors described in this article, you can easily integrate floating-point arithmetic into Forth applications.

DIE AUFGABE

Im letzten Heft ging es um formatierte Zahlenausgabe. Diesmal soll Text formatiert ausgegeben werden. Die Aufgabe baut unmittelbar auf 'immediat comments' auf - Kommentaren, die ausgegeben werden, noch während ein File geladen oder interpretiert wird. Benutzer des volksForth83 oder des F83 werden das Wort .(kennen. Es ist IMMEDIATE und gibt den in einer Quelle folgenden String als Text auf den Bildschirm aus. Damit lassen sich Bedienungsanweisungen an Benutzer elegant in Files ablegen. Das gesuchte Wort .FORMAT(geht noch einen Schritt weiter. Es soll den Programmierer durch eine formatierte Ausgabe auch längerer Texte besser unterstützen. Die Aufgabe ist wie immer als Glossar des Wortes formuliert. Die volksForth83- und die F83-Quellen können selbstverständlich genutzt werden. Entsprechende Auszüge der Listings schickt die Redaktion gern zu. (Bitte einen frankierten Rückumschlag beilegen. Danke.) Viel Spaß beim tüfteln. (mk)

```
.FORMAT(     ( Text ( -- )    immediate
Druckt den nachfolgenden Text des Input-Stream bis zum Delimiter
formatiert aus, d.h. ein Wort im Text wird am Anfang der nächsten
Zeile ausgegeben, wenn es über den rechten Rand hinausgehen würde
(word-wrapping). Die schließende runde Klammer ist der Delimiter.
Benutzt die Werte der Variablen LMARGIN für den linken, RMARGIN
für den rechten, TMARGIN für den oberen und BMARGIN für den
unteren Rand und L/PAGE für die Länge einer Seite gerechnet in
Zeilen.
```

```
Scr # 5          TEST.BLK
  0 \ Ja, was ist denn das?
  1 : A   ascii a emit r> ;
  2 : B   ascii b emit    ;
  3 : C   ascii c emit    ;
  4 : D   ascii d emit >r ;
  5
  6 : AB  a b ;
  7 : CD  c d ;
  8
  9 : ABCD  ab cd ;
 10
 11 \S   Raten Sie mal:
 12      Was wird von ABCD, was von AB CD und was von A B C D
 13      ausgegeben? Wieso eigentlich? Und kann man solche Effekte
 14      sinnvoll nutzen?
 15      Auf Antworten freut sich Ihr FORTH MAGAZIN.
                                                        (mk)
```

DAS FORTH BÜRO
der
Forth Gesellschaft eV

Friedensallee 92, 2000 Hamburg 50
Tel: 040-3904204 jeden Dienstag von 18:00-20:00 Uhr

Postgiroamt Hamburg, Kto: 563211-208, BLZ 20010020

GRUPPEN DER FORTH GESELLSCHAFT eV, BRD

6100 Darmstadt: Andreas Soeder, 06257-2744
Treffen an der VHS an einem Mittwoch in der Mitte des Monats.
2000 Hamburg: (über das FORTH BÜRO, 040-3904204 Di.18-20Uhr)
Treffen jeden vierten Samstag im Monat ab 16:00Uhr in der Berufsfachschule für Radio- und Fernsehtechnik, Eimsbüttelerstr.64-66.
8000 München: Heinz Schnitter, 089-3103385
und Christoph Krinninger 089-7259382
Treffen jeden vierten Mittwoch im Monat 19:30Uhr im Vereinsraum 1 im Bürgerhaus Unterschleißheim am Rathausplatz (S-Bahnhaltepunkt S1 Unterschleißheim)
5600 Wuppertal: Michael Kalus, 02336-82204
Treffen jeden vierten Freitag im Monat ab 20:00Uhr im Bahnhof Ottenbruch, Funckstrasse, W'tal-Elberfeld.

FORTH INTEREST GROUPS in EUROPE

-Belgien FIG Chapter: Luk van Loock, Tel: 03-658-6343
Lariksdreff 20, B-2120 Schoten
Und: Jean-Marc Bertinchamps, Tel: 071-213858
Rue N. Monnom, 2, B-6290 Nalinnes
-Englisches FIG Chapter: Keith Goldie-Morrison
Bradden Old Rectory, Towchester, Northhamptonshire, NN128ED
-Frankreich FIG Chapter: Jean-Daniel Dodin, Tel: (16-61)44-03
77 Rue du Cagire, F-31100 Toulouse
Und: Petremann, Association Jedi,
8, Rue Pourier de Narcay, 75014 Paris
-Hamburg FIG Chapter: Siehe oben unter FORTH BÜRO.
-Holland FIG Chapter: Adriaan van Roosmalen, Tel: 31-76-713104
Heusden Houtsestraat 134, NL-4817 We Breda,
-Irland FIG Chapter: Hugh Doggs, Tel: 051-75757 od. 051-74124
Newton School, Waterford
-Italien FIG Chapter: Marco Tausel, Tel:02-645-8688
Via Gerolamo Forni 48, I-20161 Milano
-Schweiz FIG Chapter: Max Hugelshofer, Tel: 01-833-3333
Stationsstrasse, CH-8306 Bruttisellen,
Und: Renato Mauerhofer, Cassinelle 17, CH-6982 Agno

EUROFORML

FORTH MODIFICATION LABORATORY

(FORML)

EUROPA FORML KONFERENZ 1987

EuroFORML Konferenz
18 - 20 September 1987
Schloß Stettenfels, Bundesrepublik Deutschland

Veranstalter
"Forth Gesellschaft eV, BRD"
"Forth Interest Group Inc, USA"

EINLADUNG

und Aufruf, Arbeiten einzusenden zu den Themen:

Programmiersprache FORTH
und
FORTH Prozessoren

(Ausführliche Beschreibung im Heft)

Anfragen, Reservierungen für die Konferenz und die Arbeiten richten Sie bitte an:

C.D. Osten
Gneisenaustr. 23 / D-2000 Hamburg 20 / FRGermany
(49) (40) 422 1694 or (49) (40) 490 5195