1986

VIERTE DIMENSION

Vierte Dimension
Volume II/Nr. 4

FLOATING POINT
ALERT BOX
TIPKURS
DECOMPILER NUCLEUS
„GEMEINNÜTZIG"

FORTH MAGAZIN

5,00 DM

ANZEIGE

(Andreas Goppold)

****

P R O F E S S I O N E L L E

Forth-Programmierer gesucht

Fuer anspruchsvolle Aufgaben im Bereich Forth-Systeme-Entwicklung und Anwendungen suchen wir immer Forth-Programmierer, die die professionelle Arbeitsweise beherrschen. Das bedeutet fuer uns: Nicht nur genialer Code, sondern auch lesbar, mit allgemein verstaendlicher Dokumentation. Bereitschaft und Faehigkeit, sich in den Code anderer Programmierer einzuarbeiten. Termingerechtes Arbeiten, Faehigkeit, den Aufwand fuer ein Projekt richtig einzuschaetzen. Und dann, auch: gute Forth-Programmierkenntnisse.

Aufgaben: Forth-Weiterentwicklung (Workstation-Forth), Forth (Target)-Compiler, sowie die ueblichen Programmierjobs.

Arbeitsbedingungen: Flexibel (Freiberuflich, Angestellt, und Teilhaberschaft sind moeglich)

Bezahlung: Nach professionellen Kriterien. Leistungs- und Erfolgs-orientiert.

Lernmoeglichkeit: Sollte sich jemand von obigen hohen Zielen angesprochen fuehlen, aber seine eigenen Faehigkeiten noch nicht so hoch einschaetzen, so ist die Moeglichkeit der Ausbildung gegeben.

Wer sind wir? Andreas Goppold und einige Mitstreiter (Computer Gilde)

Wo sind wir? Mittlerweile quer ueber ganz Deutschland.

Bitte schreiben an:
Andreas Goppold
Wassermannweg 18
2Ø5Ø Hamburg 8Ø

oder

Andreas Goppold
Postlagernd
7869 Schoenau (Schwarzwald)
Tel. Ø7673-1Ø79

Fuer Mailboxer:
Delphi: Datex-P: Ø31256 17Ø3Ø88 ANDROMEDASG
MicroModul: Datex-P: 45667313340 ANDROMEDASG

CODE: Wer professionell geschriebenen Forth-Code professionell vermarkten moechte, moege sich bitte ebenfalls an uns wenden.

****

STATT EDITORIAL

Es war einmal zur Zeit t=0 ein armer, aber rechtschaffener Vierpol namens Eddy Wirbelstrom. Er bewohnte einen bescheidenen möblierten Hohlraum mit Dielektrikum und fließend kaltem und warmem Sättigungsstrom. Leider mußte er in der kalten Jahreszeit für die Erwärmung der Sperrschichten noch extra zahlen. Seinen Lebensunterhalt bestritt er mit einer Transduktorverstärkung.

Eddy liebte mit der ganzen Kraft seiner Übergangsfunktion Ionchen. Ionchen, die induktive Spule mit dem kleinsten Fehlwinkel im ganzen Kreise und die Tochter der einflußreichen EMK. Ihr remanenter Ferritkörper, ihre symmetrischen Impedanzen und ihre überaus harmonischen Oberwellen brachten auch schon ausgediente Leydener Flaschen zu Überschlägen im Dielektrikum (was viel heißen will)!

Ionchens Vater, Cosinus Phi, ein bekannter Industriemagnet und Leistungsfaktor hatte allerdings schon konkrete Schaltpläne für die Zukunft seiner Tochter. Sie sollte nur einer anerkannten Kapazität mit ausgeprägtem Nennwert angeschlossen werden. Aber wie so oft, der Zufallsbetrieb wollte es anders.

Als Ionchen eines Tages mit ihrem Mikrofahrrad vom Friseur nach Hause fuhr - sie hatte sich eine neue Sinushalbwelle legen lassen - da geriet ihr ein Sättigungszahn in die Filterkette. Aber Eddy Wirbelstrom, der die Gegend frequentierte, eilte mit minimaler Laufzeit hinzu, und es gelang ihm, Ionchens Kippschwingung noch vor dem Maximum der Amplitude abzufangen und gleichzurichten.

Es ist sicher nicht dem Zufall zuzuschreiben, daß sie sich bald wiedersahen. Eddy lud Ionchen zum Abendessen ins "Goldene Integral" ein. Aber das Integral war bekanntlich geschlossen. "Macht nichts", sagte Ionchen, "ich habe zu Mittag fast 0,2 Kilo-Hertz gegessen und die Sättigungsinduktion bis jetzt gehalten und außerdem muß ich auf meine Feldlinien achten". Unter irgendeinem Vorwand lud Eddy daraufhin zu einer Rundfahrt im Rotor ein. Aber Ionchen lehnte ab: "Mir wird bei der zweiten Ableitung immer so übel". Und so unternahmen sie, ganz entgegen den Schaltplänen von Vater Cosinus Phi, einen kleinen Frequenzgang ins naheliegende Streufeld.

Der Abend senkte sich über die komplexe Ebene und am Himmel erglänzten die Sternschaltungen. Nur ein einsamer Modulator flog vorbei, sanft plätscherten die elektromagnetischen Wellen und die Röhren rauschten leise. Bei der Wheatstoneschen Brücke genossen Eddy und Ionchen innig die leitende Verbindung.

Und wenn sie nicht gedämpft wurden, dann schwingen sie noch heute .....

+++ AKTUELLE NACHRICHTEN +++

Am 21.11-23.11.86 war das Jahrestreffen der Forth Gesellschaft in Bispingen in der Lüneburger Heide. Leider waren nur 14 Mitglieder der FG dabei. Die Mitgliederversammlung war damit nicht Beschlußfähig. Die Gruppe der 14 hat daraufhin beraten und Empfehlungen formuliert, die den Mitgliedern nun zur Abstimmung per Briefwahl zugehen werden, damit es im das Geschäftsjahr 1987 weitergehen kann. Die Gesellschaft hat sich von dem Zusammenbruch des Büros in diesem Frühjahr erholt. Das Büro arbeitet wieder zuverlässig. Auch das Direktorium ist in der Zusammensetzung Pauck, Schleisiek, Storjohan erfolgreich in der Geschäftspolitik tätig gewesen. Doch der 1985 begonnene Aufbau des guten Rufes und des Bekanntheitsgrades hatten in der ersten Jahreshälfte wohl gelitten. Trotz erfreulich steigender Zahl der neuen Mitglieder in der zweiten Jahreshälfte (volksFORTH-Effekt) ist die Gesamtzahl der Mitglieder mit 190 auf dem Stand von Ende 1985. Es gab nur wenige spontane Spender. Publikumswirksame Veranstaltungen gab es keine, da an der inneren Struktur der FG gearbeitet wurde. Der TREE läuft, das Büro funktioniert wieder und die Zeitung erscheint jetzt regelmäßig. Nun gilt es deutlich mehr Mitglieder zu gewinnen und das Spendenaufkommen zu erhöhen, damit der Verein in die Lage versetzt wird, zeigen zu können, welchen Beitrag er in der programmierenden öffentlichkeit leisten kann. Und dabei geht neben der Vermittlung von Basiswissen um die Ausbildung von Urteilskraft. Die Programmiersprache Forth ist dazu besonders geeignet, weil sie Einsicht gewährt. 1987 soll für die FG wieder geworben werden.

volksFORTH-83 update 3.8 Austauschaktion angelaufen im November für den ATARI 520ST. Alle User bekommen jetzt ihre neue Fassung bei: Bernd Pennemann, Steilshooperstr.46, 2000 Hamburg 60, 040-6900539.

volksFORTH-83 3.8 für SCHNEIDER CPC 464/664 wurde soeben fertig. Ebenfalls ab sofort verfügbar bei Bernd Pennemann, zwei Disketten 3" mit System und Quellen sowie Handbuch für 55.-DM.

volksFORTH-83 für CP/M 2.2 verfügbar auch auf 5.25" Disketten. Muttermaschine ist der ITT 3030. Werden 2 formatierte Disketten abgegeben, können andere CP/M 2.2 Clones erzeugt werden. Bitte auch an Bernd Pennemann wenden. Mit Handbuch wohl auch 55.-DM. Noch im Dezember soll's auch im AppleII-CP/M2.2-Diskdrive-Format vorliegen.

INHALT

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

Die Arbeiten sollten folgendes enthalten:

1. TITELSEITE mit dem Titel, Autor(en) und Institut(en) oder Bertieb(en), in denen sie angefertigt wurden.
2. Eine ZUSAMMENFASSUNG von 50-100 Worten, möglichst in deutscher und englischer Sprache. Die Zusammenfassung sollte Absicht, Methoden, Ergebnisse und Schlußfolgerungen der Arbeit enthalten und für sich genommen bereits verständlich sein.
3. Etwa 5 SCHLÜSSELWORTE. Um einen Index leichter erstellen zu können, werden diese mit der Zusammenfassung in beiden Sprachen wiedergegeben.
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 sollte in einem eigenen Abschnitt am Ende der Arbeit ausgesprochen werden.
6. Referenzen. Literaturstellen sollten auf einer eigenen Seite getippt sein, um sie für das Layout gesondert verkleinern zu können, sollten alphabetisch nach Autoren geordnet und durchnummeriert sein. Im Text beziehen sich die Nummern in runden Klammern auf diese Liste. Zeitungsartikel sollten 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 sollten mit dem Namen des Autors, dem vollen Titel, Ausgabe, Erscheinungsort, Herausgeber und Jahr genannt werden.
7. Illustrationen sollten 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 kontrastenreich 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. Jede Tabelle sollte 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 sollte auf einer eigenen Seite getippt sein, eine Seiten- oder Screen-Nummer und eine überschrift tragen und im Text angesprochen werden. Die Komentare 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.

Die Arbeiten sollten in doppelter Ausfertigung eingereicht werden. 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 per DFü (300Bd,ASCII) übergeben werden.

IMPRESSUM

Titel: VIERTE DIMENSION, Magazin für die Mitglieder der Forth Gesellschaft eV.
Herausgeber: Forth Gesellschaft eV, Friedensallee 92, 2000 Hamburg 50.
Forth: Klaus Schleisiek und die Mitglieder des Review-Boards sowie alle namentlich genannten Autoren.
Auflage: 500 Stück europaweit. Druck: Offsetrepro Lüdemann, Wuppertal
Erscheinungsweise: In jedem Quartal.
Redaktion & Anzeigenleitung: Michael Kalus, Präsidentenstraße 40, 5830 Schwelm, Telefon 02336-82204.
Redaktionsschluß: Der mittlere Quartalsmonat.

Das Forth Büro

Forth Gesellschaft eV, Friedensallee 92, 2000 Hamburg 50

Tel.: 040-3904204

Postgiroamt Hamburg, Kto: 563211-208, BLZ 20010020

!!! DIE FORTH GESELLSCHAFT IST EIN GEMEINNUETZIGER VEREIN !!!

P. Drechsel
Albinistr.8
65 Mainz

19.11.86

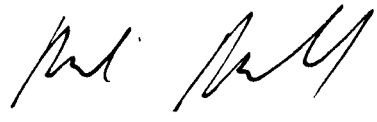
Sehr geehrter Herr Weineck

Herzlichen Dank für das Angebot des neuen volksFORTH83 Version 3.8. Dank Ihrer Großzügigkeit gerate ich jedoch allmählich in moralische Problemlagen. Da ich nachvollziehen kann, welche Arbeit in diesem Supercompiler enthalten ist, sind mir DM 10,-- Gegenleistung doch schon verwerflich gering. Andererseits bin ich mir auch nicht im Klaren darüber, ob es sinnvoll ist, jetzt einfach mehr Geld beizulegen.

Soweit ich informiert bin, dürfte doch so etwas wie ein Freundeskreis oder Ähnliches des volksFORTH83 existieren. Es würde mich interessieren, ob man hier als stiller Unterstützer beitreten kann und absetzfähige Spenden leisten kann. Informieren Sie mich doch, welche Möglichkeiten es gibt, damit ich mein schlechtes Gewissen etwas beruhigen kann.

Nochmals vielen Dank für das Update

mit herzlichen Grüßen



H E R Z L I C H G E R N E , lieber Herr Drechsel!!

Die Forth Gesellschaft ist ein gemeinnütziger Verein, stolz darauf, Herrn Weineck in ihren Reihen zu wissen und hofft, daß das volksFORTH-83 in der Public Domaine ihr gewidmet bleibt. Sie können also an die Forth Gesellschaft spenden und erhalten darüber eine Spendenquittung. Auch die stille Unterstützung ist möglich - sie können förderndes Mitglied werden. Der Beitrag ist ebenfalls absetzbar. Für welche Projekte die Forth Gesellschaft versucht, Mittel aufzubringen, können Sie im Forth Büro erfahren.

Ihr Editor Kalus

LESERBRIEFE

FORTH Coding Conventions

Wir waren angenehm überrascht von der Vielfalt der Richtlinien die Mr. Harris da zusammengestellt hat. Wir werden uns bei unserem Projekt "CAT-FORTH für den ATARI ST+" intensiv daran halten, denn wir beabsichtigen, sämtlichen Source-Code unseres Systems mitzuliefern. Da können wir diese Richtlinien sehr gut gebrauchen.
Allerdings werden wir bei unserem FORTH auf die altmodischen Screens mit je 1 kbyte Source verzichten. Und wir sehen auch keinen Grund, warum wir bei einem Rechner mit 1 Mega RAM sparsam mit dem Platz umgehen sollten. Deshalb werden wir die Richtlinien in folgenden Punkten verletzen:

1. Zwischen jedes Wort gehört eine Leerzeile'

2. Die Kontroll-Strukturen von Mr. Harris finden wir total unübersichtlich. Wir werden's wie folgt handhaben:

```
: name
   body
   IF
      body
      IF
         body
      THEN
   THEN
   body ;
```

Das braucht zwar einiges mehr an Platz, ist aber viel übersichtlicher.

Wir würden uns freuen, wenn die "Vierte Dimension" dieses Thema aufgreifen würde, sicher gibt es noch viele Lösungen dazu.

Erich Suter, Hauptstrasse 46, CH-6436 Muotathal, Schweiz

FIG oder FG?

Die Idee, in der VD aus der FORTH DIMENSIONS zu referieren, finde gut. Aber noch weit besser fände ich, wenn ihr die FORTH DIMENSIONS direkt nachdrucken würdet, wie das meines Wissens die FIG ITALY in Mailand auch macht. So bin ich Mitglied in der FIG statt in der FG. K.P.Schleisiek, Aachen
(FORTH MAGAZIN: Die Forth Gesellschaft versucht derzeit ein Kooperations-Abkommen mit der FIG in USA zu treffen. Darin soll ein Literaturaustausch vereinbart werden, um die Dollarbarriere abzubauen. In der Redaktion sind alle Artikel aus der Forth Dimensions erhältlich. Mit der Rubrik AKTUELLE LITERATUR wird ab sofort ein Überblick geboten. Der Kopierservice der Forth Gesellschaft kann darüberhinaus fast alles in Forth Publizierte irgendwie beschaffen. Ausgeschlossen davon sind komplette Bücher.)

Wo sind die Anzeigen der Anbieter???

Ich suche für eine Amateurfunk-Anwendung ein 6502-Einkartenrechner mit dem dazugehörigen angepassten Forth für einfache Steueraufgaben, anzusprechen über eine RS232 oder IEC-BUS. Verfüge über CBM 8032 mit Floppy 8050 und SINCLAIR SPECTRUM. Rudolf Lohmer, 5480 Remagen, 02642-3203 od 0228-135458.

Forth für Commodore C-16, C-116 und Plus/4

Wer hat Lust, mit mir zusammen eine Forth-Implementierung für diese Rechner zu fabrizieren oder sitzt womöglich schon am gleichen Projekt? Commodore hat diese Rechner mittlerweile ca. 1 Million mal verkauft. Außer Basic gibt es dafür keine höhere Programmiersprache. Wer mit mir in die Geschichte eingehen will, wende sich an: Claus Vogt, 1 Berlin 30, Tel: 030-2168938
(FORTH MAGAZIN: Da gibts nur eins: volksFORTH-83 hochziehen!!! Einweisung gibt Bernd Pennemann oder Ulrich Hoffmann. Forth Büro fragen.)

Der Atari volksFORTH-83 GEM Editor von Dietrich Weineck

Lieber Dietrich, ich habe vor einigen Tagen bei einem Bekannten auf dessen Atari 520ST deinen Editor zum volksFORTH-83 ausprobieren können. Das ist das tollste, was ich seit langem gesehen habe - der Editor ist absolute erste Sahne! Klasse, Mann. Phantastisch! Wenn's das auch für den Amiga gäbe. (mk)

Erratum

In VD86II/3 im Artikel 'Queues in Forth' muß es auf S.27 heißen "...folgenden NEUN Worten..." statt SECHS. Auf S.11 im Scr#1 fehlt in BM2 die schließende LOOP.

PREISRÄTSEL

Auflösung aus dem letzten Heft: Das Wort heißt WURZEL. Geschrieben haben uns:

(1) Finn Berlev, Lillevangsvey 92, DK-3520 Farum, Dänemark
(2) Ulrich Hoffmann, Harmsstrasse 71, D-2300 Kiel 1, BRD
(3) Joh. Polster, Speerstrasse 7, CH-8820 Wädenswil, Schweiz

Herzlichen Glückwunsch im Namen der Forth Gesellschaft. Die zugrunde liegende Formel lautet: (n+1)^2 = n^2 + (2n+1)
Herr Polster kommentierte den Stack und den Algorhythmus treffend als: WURZEL (Quadratfläche -- Seitenlänge)
Herr Hoffman hatte gleich eine Handvoll Namen für die Funktion beizutragen. Er schrieb: "Der offensichtlichste war SQR, aber Pascal meint lieber SQRT. Ich persönlich tendierte lange zu etwas wie CRT (sprich: carrot) oder in gut deutsch MHRB (sprich: Mohrrübe)" und schuf schließlich für die Ikonen-Fans v- als Namen. Und er wies darauf hin, daß am Ende der do-loop immer ein ungerader Wert auf dem Stack liegt und damit das folgende 1- einfach weggelassen werden kann, da die Hälfte auch so richtig berechnet wird, integer natürlich.

Herr Berlev verkürzte den Code SQRT auf elegante Weise. "This problem made me think on times long ago, when I learned how to do this calculations by hand..." und er fügte DSQRT hinzu " ...not as elegant, but it can be used for d-numbers and also finds the remainder."

```
: SQRT  ( limit -- n )   1 swap 0 ?do 2+ dup +loop 2/ ;

: (DSQRT1)   ( ud -- a0 a1..an - 1 n )
   1 begin >r 4 ud/mod 2dup or while r> 1+ repeat  2drop 1- r> ;
: (DSQRT2)   ( a0 a1..an - 1 n -- ur uq )
   1 under ?do
     2* -rot 4 * + over 2*
       2dup u> if   - 1- swap 1+
              else drop swap      then loop ;
: DSQRT   ( ud -- ur uq ) 2dup d0= ?exit (dsqrt1) (dsqrt2) ;
```

Glossar

SQRT n1 -- n2
"squareroot"
Nimmt einen Wert n1 vom Stack und berechnet die Quadratwurzel n1 als ganzzahligen Wert ohne den Rest.
DSQRT ud -- ur uq
"d-squareroot"
Nimmt eine doppeltgenaue Zahl ohne Vorzeichen vom Stack und berechnet deren Quadratwurzel uq und den Rest ur sodass ud = uq^2 + ur gilt.

Die neue Aufgabe

Und nun eine neue Aufgabe für die Forth Gemeinschaft Europa. Diesmal sind der Name und die Funktion bekannt gesucht wird nach den elegantesten Lösungen des Problems. Die Aufgabe ist als Glossar zum Wort gestellt. Das Wort INPUT# soll gebaut werden. Die Lösung muß frei sein von maschinenbezogenen Tricks. Schicken sie ihre Lösung an das Forth Büro in Hamburg zu Händen von Ulrich Hoffmann. Das FORTH MAGAZIN wünscht viel Vergnügen.

Aufgabe Nr.2

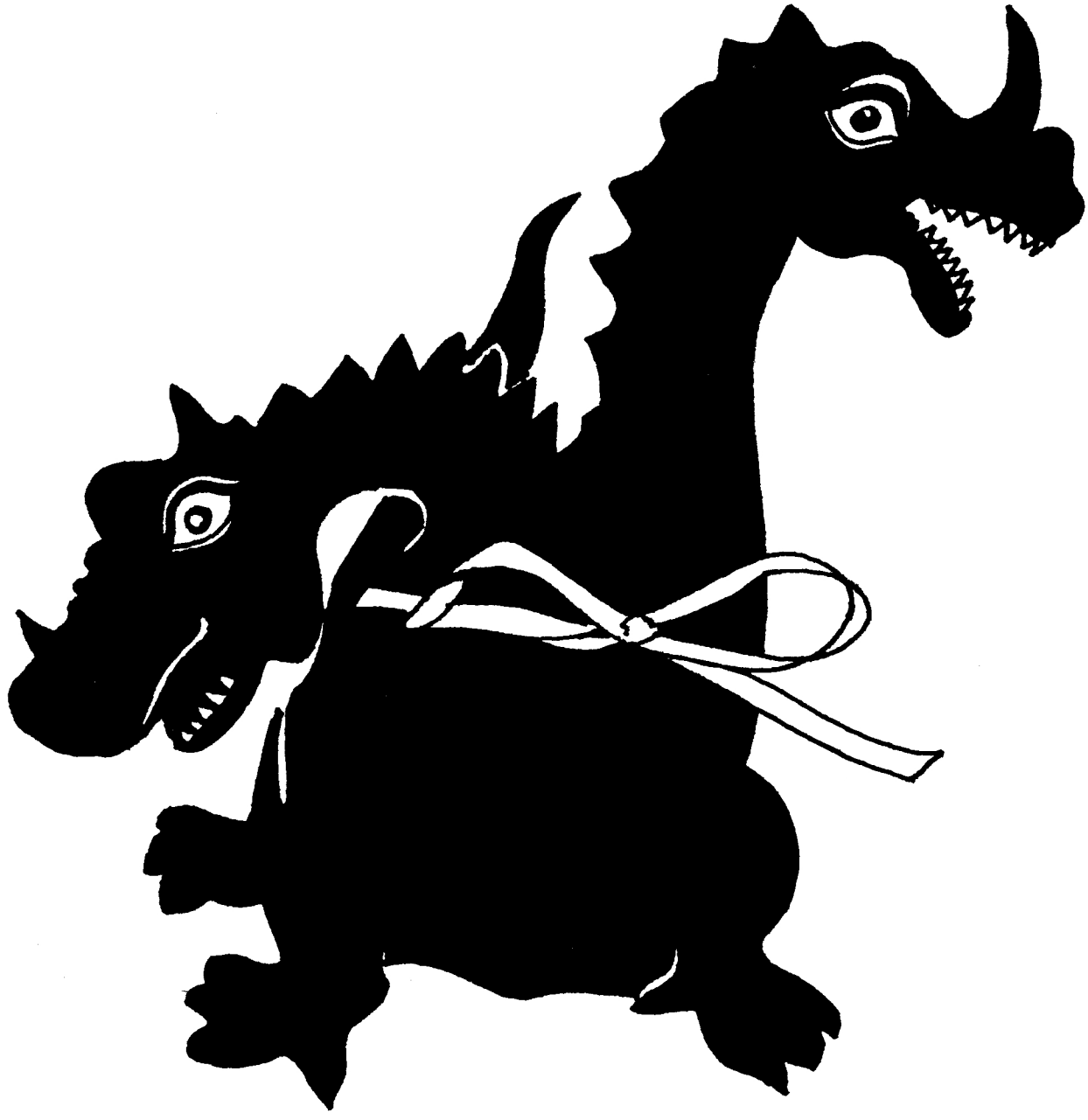
INPUT# n -- d "input-numerisch"
Numerische Eingabe, die einen Ziffernstring von maximal n Ziffern als Eingabe erwartet und diesen in die doppeltgenaue Zahl d umwandelt. Akzeptiert als Zeichen nur die Ziffern 0 bis 9 sowie die Sonderzeichen 'Komma', 'Punkt', 'Plus' und 'Minus'. Die Eingabe bleibt edierbar, bis sie mit <return> abgeschlossen wird.

AUS DEM VEREIN

Der Pieris Maidaia Preis

In diesem Jahr war Pieris Maidaia vier Jahre alt. Er lebt in den USA und hatte einen zweiköpfigen blaugrünen geschuppten Drachen mit großen rot-funkelnden Augen und großen Mäulern. Diesen Gummi-Drachen - made in Honkong - mit den zwei Köpfen, der faucht wenn man ihn drückt, hat er aus seiner Spielkiste heraus der Forth Gemeinschaft in Europa gestiftet. Und es wurde der SWAP-DRACHEN daraus, der Preis der Forth Gesellschaft für besondere Verdienste um die Forth Gemeinschaft in Europa. In nun, 1986, im zweiten Jahr der FG, wurde dieser Preis zum erstenmal vergeben. Noch ist der Pieris Maidaia Preis nicht mit einer Prämie verbunden. Doch wir hoffen alle, daß sich für 1987 ein Mäzen (freigiebiger Gönner) findet, der diesen Preis mit 1000.-DM dotiert. Und daß auch in den nächsten Jahren der Preis wieder mit diesem Wert ausgestattet werden kann.

Auf dem Mitglieder-Jahrestreffen 1986 in Bispingen ging der Preis - spontan und noch ohne Prämie - an Michael Kalus, dem Editor des FORTH MAGAZIN als Anerkennung für seine ehrenamtliche Arbeit an unserer Zeitung und in der FG. Er hats verdient!



LITERATURSCHAU

TABLE OF CONTENTS

FORTSETZUNG S.30

FIG GESCHICHTE

FIG-Forth ist die aelteste aller Public Domain Implementationen.
Die Forth Interest Group (FIG) wurde 1978 in Californien, USA geruendet.
Sie bestand aus einer Handvoll Programmierern, die der Ansicht waren, dass Forth ein entscheidender Fortschritt in der Programmierung von Computern, speziell Microcomputern ist, und wollten es daher einer groesseren Anzahl von Programmierern zugaenglich machen. Denn bis dahin war Forth nur wenigen Insidern bekannt, und bestehende Implementationen wie Polyforth zeichneten sich neben einer hohen Leistungsfaehigkeit auch durch einen ebenso hohen Preis aus.
Die FIGger erkannten, dass die einzige Moeglichkeit, Forth zu verbreiten, darin bestand, eine durchschaubare, preiswerte Implementation auf einer Vielzahl von Rechnern zu realisieren.
Also entschloss man sich, ein gemeinsames Forth auf allen gaengigen CPUs zu implementieren und die Sourcelistings in die Public Domain zu geben, das heisst, sie jedermann ohne Lizenzgebuehren kostenlos zur Verfuegung zu stellen. Es entstanden also Assembler Listings fuer folgende Rechnertypen :

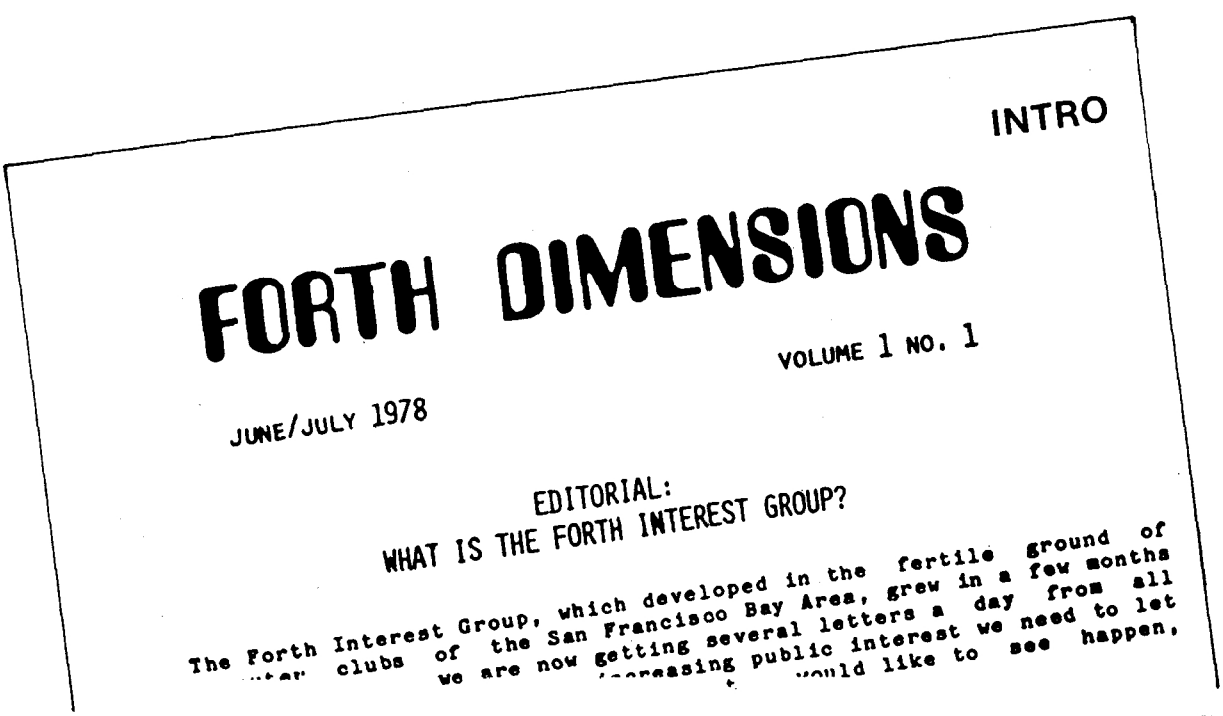
1802 (3/81), 6502 (9/80), 6800 (5/79), 6809 (5/80), 8080 (9/79),
8086/8 (3/81), 9900 (3/81), ALPHA MICRO (9/80), APPLE II (9/80),
ECLIPSE (10/82), IBM PC (3/84), NOVA (5/81), PACE (5/79),
PDP-11 (1/80), VAX (10/82), Z80 (9/82)

(Datum der Herausgabe in Klammern.)
Ebenso wichtig wie die Veroeffentlichung der Assemblerlistings war die Herausgabe der Zeitschrift "Forth Dimensions". Durch sie wurde ein reger Informations- und Erfahrungsaustausch in Gang gesetzt.
Ferner wurde das Forth Standards Team (FST) gegruendet, das auf den alten FIG Implementationen basierend, die neuen Standarts Forth 79* und Forth 83 entwarf. Neue Ideen werden jaehrlich auf der FORML (Forth Modification Laboratory) Konferenz diskutiert und in Form der FORML Proceedings auch Nicht-Konferenzteilnehmern zugaenglich gemacht. Die Forth Gesellschaft konnte im Oktober 85 die euroFORML'85 als erste bundesdeutsche Forth Konferenz organisieren, an der Forthler aus ganz Europa und den USA teilnahmen.

Die Adresse der Forth Interest Group ist :

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155
(408) 277-0668

Die Mitgliedschaft bei FIG USA kostet $27 bzw. $33 (Luftpost). Man erhaelt damit automatisch die Zeitschrift "Forth Dimensions".



INTRO

FORTH DIMENSIONS

VOLUME 1 NO. 1

JUNE/JULY 1978

EDITORIAL:
WHAT IS THE FORTH INTEREST GROUP?

The Forth Interest Group, which developed in the fertile ground of ...ter clubs of the San Francisco Bay Area, grew in a few months we are now getting several letters a day from all ...reasing public interest we need to let ... would like to see happen,

DIE KUNST, NAMEN ZU GEBEN

Klaus-Peter Schleisiek , Aachen

1946 kam ich zur Welt und 1978 zum Diplom der Elektrotechnik der TH Aachen. Hier bin ich auch zur Zeit (wieder) tätig. FORTH ist für mich ein Hilfsmittel, um Einkartenrechner zu programmieren.

Henry Laxen schrieb in CHOOSING NAMES, daß gute Namen das wichtigste sind, und daß die Namenswahl den Unterschied ausmacht zwischen lesbaren und unlesbaren Programmen. Dem stimme ich zu. Auch teile ich seine Ansicht, daß Stack-Operatoren nicht unbedingt kleine Pfeile haben müßten. Schade, daß im Standard-Wortschatz die Pfeile benutzt werden, zB für den Datentausch mit dem Return-Stack oder beim Formatieren der Zahlenausgabe (Fußnote1).

Einige meiner Namens-Konventionen möchte ich hier zur Diskussion stellen. Dabei möchte ich an bekannten Worten zeigen, wie ihre Funktion klarer ausgedrückt werden könnte.
Ich schreibe statt >R R> R lieber RPUSH RPULL RCOPY und nehme zB UPUSH UPULL UCOPY als Namen für User-Stack-Operationen.

An die Postfix-Notation in FORTH habe ich mich leicht und gerne gewöhnt. Einige Worte aber beziehen sich nicht auf Vorhergehendes, sondern auf das im Text f o l g e n d e Wort. Diese wichtige Ausnahme verlangt geradezu nach Kennzeichnung - der schließende Doppelpunkt bietet sich an. Ich schreibe statt CODE CONSTANT USER VARIABLE VOCABULARY lieber CODE: CONST: UVAR: VAR: VOCABULARY: und dann zB ARRAY: VEKTOR: ATTRIBUT: RELATION: für Defining Words.

Leo Brodie (2) hat das schließende Fragezeichen für Test-Worte reserviert - FERTIG? KEY? - die ein Flag auf dem Stack hinterlassen.

Weil ich die Umschalt-Eigenschaft einiger Worte sehen möchte, gibt es bei mir mangels Backslash das vorangestellte 'Prozentzeichen'. Ich schreibe statt TOGGLE SMUDGE IMMEDIATE lieber %TOGGLE %SMUDGE %IMMEDIATE .

```
( Zum Ausprobieren, Rockwell-figFORTH )
: RPUSH   ?COMP COMPILE >R ;  IMMEDIATE      : %TOGGLE TOGGLE ;
: RPULL   ?COMP COMPILE R> ;  IMMEDIATE      : %SMUDGE SMUDGE ;
: RCOPY   ?COMP COMPILE R  ;  IMMEDIATE      : %IMMEDIATE  IMMEDIATE ;

: CODE:   ACOMPILEÜ CODE ;  IMMEDIATE
: CONST:  CONSTANT ;
: UVAR:   USER SMUDGE ;
: VAR:    VARIABLE ;
: VOCABULARY:  VOCABULARY ;
```

(1) Henry Laxen, FORTH DIMENSIONS Vol.IV/4, S.33ff
(2) Leo Brodie, Programmieren in Forth, Hanser, 1984
(3) Kim Harris, Forth Coding Conventions, Asilomar FORML Conference 1985, S.143ff

(FORTH MAGAZIN: Gute Namen sind bei Forth das Salz in der Suppe. Aber noch wichtiger ist STYLE. Das meint die ganze Art und Weise der Documentation des Code. Und hier vor allem die Kommentierung des Geschehens auf dem Stack. Eine ausführliche Übersicht der Konventionen hierzu - aus der Praxis geschöpft und aufbereitet - gibt Kim Haris (3).

Fußnote1: Der Standard ist hier inkonsistent. Mal wird die Winkelklammer zur Anzeige einer Richtung und mal als Einklammerung einer Phrase benutzt. So meint R> eine Richtung und in <# # # #S #> wird eine Phrase in die Klammern eingebettet - es gibt das einleitende Wort <# und das Wort #> als Abschluß. Der Pfeil als Richtungsanzeiger in Worten kommt langsam aus der Mode, da er 'kryptographisch' und wenig selbstverständlich ist.)

Defining Words II

Konrad Scheller, Forchheim

Nun wollen wir uns wieder einmal den Defining Words zuwenden. Letztesmal haben wir ja nur die Grundlagen behandelt, nun aber kommen die Anwendungen.

Zuerst nur mal *das* Standardbeispiel, das einem man fast überall begegnet, wenn man etwas Forth-Literatur liest. Das Wort "Emitter" baut neue Worte, die eine Reihe von ASCII-Zeichen ausgeben. Im Grunde handelt es sich dabei um eine Verallgemeinerung. In Leo Brodie's legendärem Buch "Startin Forth" trifft man auf dieses Beispiel:

```
: Stars   ( n -- )
  0   ?DO   ASCII * Emit   LOOP   ;
```

(Ich habe dieses Beispiel etwas an Forth83 angepasst). Nun könnte man also, hätte man nicht nur Sterne, sondern z.B. auch Striche --- oder Unterstreichungen _____ oder etwa Punkte .... zu drucken, für jedes Zeichen ein eigenes Wort definieren. Blödsinn! Dafür gibt es Defining Words:

```
: Emitter  ( char -- )
  Create   C,                        \ kompiliert Zeichen ein.
  Does>    ( n -- )                  \ Die "pfa" wird nicht
           C@                        \ angegeben!
           swap 0 ?DO  dup Emit  LOOP  drop ;
```

Und nun:

```
32 Emitter Spaces            ASCII * Emitter Stars
ASCII - Emitter Hyphens      ASCII _ Emitter Underlines
ASCII . Emitter Dots         usw,usw..
```

Nun wird es interessant: Datenfelder. Die klassische Methode zur Implementierung eines Datenfeldes in Forth sieht so aus:

```
Create Groessen   20 Allot      ( Feld Groessen, zehn 16-Bit-Werte)

: Groesse      ( Groesse -- addr )      2*  Groessen + ;
```

Solange man nur ein oder zwei Felder braucht, lohnt es sich natürlich nicht. Interessant wird es ab so drei bis vier Feldern die man braucht:

```
: Feld          ( max.Groesse -- )
  Create   dup ,    2*  allot      \ Groesse einkompilieren, Raum
  Does>    ( n -- addr )           \ Fehlererkennung eingebaut!
           2dup @ < not Abort" Range Error"
           swap 2* + ;

10 Feld Groessen          10 Feld Umsatz          10 Feld Verbrauch
```

Hier spart man sich Code, und vor allem Entwicklungszeit.
Wird später z.B. die Grösse von 16 bit auf 8 oder 32 bit geändert, dann hat man nur ein einziges anstatt drei oder vier Wote zu ändern.

Oft braucht man auch zweidimensionale Felder. Nun will ich aber nicht beide Versionen vorstellen (ohne und mit Defining Words), sondern ich gehe davon aus, dass Sie das Grundsätzliche bereits intus haben und zeige deshalb nur die "bessere".

```
: Feld  ( max.x   max.y  -- )
  Create  over , dup ,   * Allot      \ Speicher reservieren
  Does> ( x y -- addr )   >r          \ pfa auf ReturnStack
        dup  r@ 2+ @ < not Abort" Y-Range Error"
        over r@    @ < not Abort" X-Range Error"
        r@ @ *              \ hole max.x und nimm es mit y mal....
        + r> + ;            \ zähle x und die pfa dazu.
```

Ganz allgemein kann man zu den oben gezeigten Beispielen noch sagen: Die Fehlerbehandlung sollte man nur in der Testphase beibehalten und später, beim fertigen Programm, herausnehmen. Die Programme laufen dadurch schneller. Ausnahme: Wenn es dem User möglich ist, Fehler zu machen, indem er zu grosse oder zu kleine Indizies eingibt.

Zum Schluss möchte ich noch ein Beispiel anbringen, das nicht von mir ist. Es stammt von Leo Brodie aus seinem Buch "Thinking Forth". Es gefiel mir so gut, dass ich es einfach bringen muss.

```
HEX  01A0  CONSTANT  BASIS.PORT.ADDRESS
     BASIS.PORT.ADDRESS     CONSTANT SPEAKER
     BASIS.PORT.ADDRESS 2+  CONSTANT FLIPPER-A
     BASIS.PORT.ADDRESS 4 + CONSTANT FLIPPER-B
     BASIS.PORT.ADDRESS 6 + CONSTANT SPOTLIGHT
DECIMAL
```

Das war die ursprüngliche Version. Leo entwickelte nun dieses Stück Code immer weiter - was sehr aufschlussreich war - und am Schluss stand diese Version:

```
: PORT     ( 'port -- 'next-port )    dup Constant  2+  ;
      \ Does>  ;    ( -- 'port )
  HEX  01A0   ( BasisPortaddr )
       PORT SPEAKER
       PORT FLIPPER-A
       PORT FLIPPER-B
       PORT SPOTLIGHT
  drop DECIMAL
```

Hier ist noch eine Möglichkeit wahrgenommen, die Forth uns bietet. Statt CREATE wurde CONSTANT verwendet, was zwei Vorteile hat. Erstens sparen wir uns "Create ," denn nichts anderes ist ja Constant. Zweitens ersetzt der Laufzeitcode von Constant uns das "Does> @" Dadurch geht es schneller und verbraucht weniger Speicher.

[1] Leo Brodie, Starting Forth. Prentice Hall
Zu beziehen bei Forth Systeme Flesch, Postfach 1226,
7829 Titisee-Neustadt, Tel: 07651/1665

[2] Programmieren in Forth. Carl Hanser Verlag, 1984
Übersetzung von [1].

[3] Leo Brodie, Thinking Forth. Prentice Hall
Forth Systeme Flesch.

[4] In Forth denken. Carl Hanser Verlag, 1986.
Übersetzung von [3]

FORTHQUELLEN

Der folgende Beitrag lag in dem verschwenderischen Format von 50 Zeilen mit 50 Spalten vor und war dadurch 10 Seiten lang. Er mußte geschrumpft werden. Die Lesbarkeit hat dadurch gelitten. Zur Not kann eine 1:1 Kopie beim Kopierservice im Forth Büro bestellt werden.

A FAST HIGH-LEVEL FLOATING POINT

Robert F. Illyes

ISYS
PO Box 2516, Sta. A
Champaign, IL 61820
Phone: 217/359-6039

If binary normalization and rounding are used, a fast single-precision FORTH floating point can be built with accuracy adequate for many applications. The creation of such high-level floating points has become of more than academic interest with the release of the Novix FORTH chip. The FORTH-83 floating point presented here is accurate to 4.8 digits. Values may range from about 9E-4933 to about 5E4931. This floating point may be used without fee provided the copyright notice and associated information in the source are included.

FIXED VS. FLOATING POINT

FORTH programmers have traditionally favored fixed over floating point. A fixed point application is harder to write. The range of each value must be known and considered carefully, if adequate accuracy is to be maintained and overflow avoided. But fixed point applications are much more compact and often much faster than floating point (in the absense of floating point hardware).

The debate of fixed vs. floating point is at least as old as the ENIAC, the first electronic digital computer. John von Neumann used fixed point on the ENIAC. He felt that the detailed understanding of expressions required by fixed point was desirable, and that fixed point was well worth the extra time (1).

But computers are no longer the scarce resource that they once were, and the extra programming time is often more costly than any advantages offered by fixed point. For getting the most out of the least hardware, however, fixed point will always be the technique of choice.

Fixed point arithmetic represents a real number as a ratio of integers, with an implicit divisor. This implicit divisor may be thought of as the representation of unity. If unity were represented by 300, for example, 2.5 would be represented by 750.

To multiply 2.5 by 3.5, with all values representing unity as ten, one would evaluate

$$\frac{25 * 35}{10}$$

The ten is called a scale factor, and the division by ten is called a scaling operation. This expression is obviously inefficient, requiring both a division and a multiplication to accomplish a multiplication.

Most scaling operations can, however, be eliminated by a little algebraic manipulation. In the case of the sum of two squares, for example, where A and B are fixed point integers and unity is represented by the integer U,

$$\frac{A * A}{U} + \frac{B * B}{U} \quad \text{--->} \quad \frac{(A * A)+(B * B)}{U}$$

In addition to the elimination of a division by U, the right expression is more accurate. Each division can introduce some error, and halving the number of divisions halves the worst-case error.

DECIMAL VS. BINARY NORMALIZATION

A floating point number consists of two values, an exponent and a mantissa. The mantissa may represent either an integer or a fraction. The exponent and the mantissa are related to each other in the same way as the value and power of ten in scientific notation are related.

A mantissa is always kept as large as possible. This process is called normalization. The following illustrates the action of decimal normalization with an unsigned integer mantissa:

Value	Stack represention		
$5 * 10^4$	50000	0	--
$7 * 10^3$	7000	0	--
$5 * 10^3$	50000	-1	--

The smallest the mantissa can become is 6554. If a mantissa of 6553 is encountered, normalization requires that it be made 65530, and that the exponent be decremented. It follows that the worst-case error in representing a real number is half of 1 part in 6554, or 1 part in 13108. The error is halved because of the rounding of the real number to the nearest floating point value.

Had we been using binary normalizion, the smallest mantissa would have been 32768, and the worst case error in representation would have been 1 part 65536, or 1/5 that of decimal normalization. LOG10(65536) is 4.8, the accuracy in decimal digits.

As with fixed point, scaling operations are required by floating point. With decimal normalization, this takes the form of division and multiplication by powers of ten. Unlike fixed point, no simple algebraic manipulation will partly eliminate the scale factors. Consequently there are twice as many multiplications and divisions as the floating point operators would seem to imply. Due to the presence of scaling in 73% of decimally normalized additions (2), the amount is actually somewhat over twice.

With binary normalization, by contrast, this extra multiplication effectively dissappears. The scaling by a power of two can usually be handled with a single shift and some stack manipulation, all fast operations.

Though a decimally normalized floating point can be incredibly small (3), a binary normalized floating point has 1/5 the error and is about twice as fast.

It should be mentioned that the mantissa should be multiples of 2 bytes if the full speed advantage of binary normalization is to be available. Extra shifting and masking operations are necessary with odd byte counts when using the 2-byte arithmetic of FORTH.

NUMBER FORMAT AND ARITHMETIC

This floating point package uses an unsigned single precision fraction with binary normalization, representing values from 1/2 to just under 1. The high bit of the fraction is always set.

The sign of the floating point number is carried in the high bit of the single precision exponent. The remaining 15 bits of the exponent represent a power of 2 in excess 4000 hex. The use of excess 4000 permits the calculation of the sign as an automatic outcome of exponent arthmetic in multiplication and division.

A zero floating point value is represented by both a zero fraction and a zero exponent. Any operation that produces a zero fraction also zeros the exponent.

The exponent is carried on top of the fraction , so the sign may be tested by the phrase DUP 0< and zero by the phrase DUP 0= .

The FORTH-83 Double Number Extension Word Set is required. Floating point values are used with the "2" words: 2CONSTANT, 2@, 2DUP, etc.

There is no checking for exponent overflow or underflow after arithmetic operation, nor is division by zero checked for. The rationale for this is the same as with FORTH integer arithmetic. By requiring that the user add any such tests, 1) all arithmetic isn't slowed by tests that are only sometimes needed and 2) the way in which errors are resolved may be determined by the user. The extremely large exponent range makes exponent overflow and underflow quite unlikely, of course.

All of the arithmetic is rounding. The failure to round is the equivalent of throwing a bit of accuracy away. The representational accuracy of 4.8 digits will be quickly lost without rounding arithmetic.

The following words behave like their integer namesakes:

F+ F- F* F/ F2* F2/ FABS FNEGATE F<

Single precision integers may be floated by FLOAT, and produced from floating point by FIX and INT, which are rounding and truncating, respectively. DFLOAT floats a double precision integer.

NUMBER INPUT AND OUTPUT

Both E and F formats are supported. A few illustrations should suffice to show their usage. An underscore indicates the point at which the return key is pressed. PLACE determines the number of places to the right of the decimal point for output only.

```
12.34 F        F. _ 12.340
12.34 F        E. _ 1.234E1
.033 E -1002   E. _ 3.300E-1004

4 PLACES

2. F -3. F F/ F. _ -0.6667
2. F -3. F F/ E. _ -6.6667E-1
```

F and E will correctly float any input string representing a signed double precision number. There may be as many as 9 digits to the right of the decimal point. Numbers input by E are accurate to over 4 digits. F is accurate to the full 4.8 digits if there are no digits to the right of the decimal point. Conversion is slightly less accurate with zeros to the right of the decimal point because a division by a power of ten must be added to the input conversion process.

F and E round the input value to the nearest floating point value. So a sixth digit will often allow a more accurately rounded conversion, even thought the result is only accurate to 4.8 digits. There is no advantage to including trailing zeros, however. In many floating points, this extra accuracy can only be achieved by the inconvenient procedure of entering the values as hexadecimal integers.

Only the leftmost 5 digits of the F. output are significant. F. displays values from 32767 to -32768, with up to 4 additional places to the right of the decimal point. The algorithm for F. avoids double rounding by using integer rather than floating point multiplication to scale by a power of ten. This gives as much accuracy as possible at the expense of a somewhat limited range. Since a higher limit on size would display digits that are not significant, this range limit does not seem at all undesirable.

Like E input, E. is accurate to somewhat over 4 digits. The principal source of inaccuracy is the function EXP, which calculates powers of 2.

The following extended fraction is used by EXP. It gives the square root of 2 to the x power. The result must be squared to get 2 to the x.

$$1 + \cfrac{2x}{34.668 - x - \cfrac{57828}{2001.18 + (2x)^2}}$$

In order to do E format I/O conversion, one must be able to evaluate the expressions

$$10^a = 2^{a/\log_{10}(2)} \quad \text{and} \quad 2^b = 10^{b*\log_{10}(2)}$$

These log expressions may be easily evaluated with great precision by applying a few fixed point techniques. First, a good rational approximation to log10(2) is needed.

```
log10(2)       = .3010299957
4004 / 13301 = .3010299978
```

The following code will convert an integer power of ten, assumed to be on the stack, into a power of 2:

```
13301 4004 */MOD >R
FLOAT 4004 FLOAT F/ EXP
R> +
```

The first line divides the power of ten by log10(2) and pushes the quotient on the return stack. The quotient is the integer part of the power of two.

The second line finds the fractional part of the power of two by dividing the remainder by the divisor. This floating point fractional part is evaluated using EXP.

The third line adds the integer power of two into the exponent of the floating point value of the fractional part, completing the conversion.

The inverse process is used to convert a power of 2 to a power of ten.

FORTH-83 LIMITATIONS

Perhaps the most serious deficiency in the FORTH-83 standard is in the area on numeric input. In a language with extensibility as its pre-eminent feature, it is surprisingly difficult to write standard code that will alter the processing of numeric input strings by the interpreter and compiler.

It is usually a simple matter to replace the system conversion word (usually called NUMBER) with a routine of ones choice. But there if no simple standard way of doing this. The interpreter, compiler and abort language are all interwoven, and may all have to be replaced if a standard solution is sought.

This floating point package assumes that double precision integers are generated if the numeric input string contains a period, and that a word PLACES can be written that will leave the number of digits to the right of the period. This does not seem to be guaranteed by FORTH-83, although it may be safely assumed on most systems that include double precision.

If you know how this part of your system works, you will probably want to eliminate the words E and F, and instead force floating point conversion of any input string containing a period. Double precision integers could still be generated by using a comma or other punctuation.

It is suggested that future FORTH standards include the word NUMBER, which is a vector to the current input numeric word.

It is also suggested that the Double Number Extension Wordset specification include a requirement that the interpreter and compiler be able to accept input strings specifying double precision values.

COMMENTS ON THE FOLLOWING CODE

The words ". and "- leave the ASCII values for period and minus, respectively. Replace these with whatever language you prefer for insertion of ASCII values.

The size of F+ can be considerably reduced at the expense of quite a lot of execution speed. Think twice before you simplify it.

The normalizing word NORM expects the stack value under the exponent to be a double precision ~~signed~~ integer. It leaves a normalized floating point number, rounding the double precision integer into the fraction.

ALIGN and RALIGN expect an integer shift count with an unsigned double precision number beneath. They leave double precision unsigned integer results. At least one shift is always performed. RALIGN rounds after alignment.

UM/ divides an unsigned double precision number by an unsigned single precision number, and rounds the single precision quotient.

ZERO forces a floating point value with a zero fraction to also have a zero exponent.

FSIGN applies the sign of the stack value under the exponent to the exponent. The double precision integer under an exponent is left unsigned.

FEXP evaluates a power of e. It is included because it is a trivial but useful application of EXP.

GET converts the next word in the input stream into a single precision signed integer.

REFERENCES

1. Von Neumann, J., John von Neumann Collected Works, vol. 5, p.113.

2. Knuth, D. K., The Art of Computer Programming, second edition, vol. 2, pp. 238,9.

3. Tracy, M., Zen Floating Point, 1984 FORML Conference Proceedings, pp. 33-35.

THE JOURNAL OF FORTH APPLICATION AND RESEARCH

Volume 3 Number 1 1985

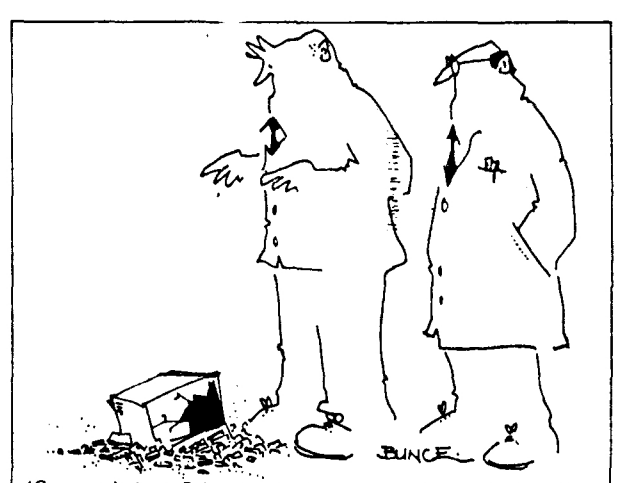

```
( FORTH-83 FLOATING POINT.
  ------------------------------------
  COPYRIGHT 1985 BY ROBERT F. ILLYES

        PO BOX 2516, STA. A
        CHAMPAIGN, IL 61820
        PHONE: 217/826-2734  )      HEX

: ZERO  OVER 0= IF DROP 0 THEN ;
: FNEGATE 8000 XOR ZERO ;
: FABS  7FFF AND ;
: NORM  >R 2DUP OR
        IF BEGIN DUP 0< NOT
           WHILE D2* R> 1- >R
           REPEAT SWAP 0< - ?DUP
           IF R> ELSE 8000 R> 1+ THEN
        ELSE R> DROP THEN ;

: F2*   1+ ZERO ;
: F*    ROT + 4000 - >R UM* R> NORM ;
: FSQ   2DUP F* ;

: F2/   1- ZERO ;
: UM/   DUP >R UM/MOD SWAP R>
        OVER 2* 1+ U< SWAP 0< OR - ;
: F/    ROT SWAP - 4000 + >R
        0 ROT ROT 2DUP U<
        IF   UM/ R> ZERO
        ELSE >R D2/ FABS R> UM/ R> 1+
        THEN ;

: ALIGN 20 MIN 0 DO D2/ LOOP ;
: RALIGN 1- ?DUP IF ALIGN THEN
        1 0 D+ D2/ ;
: FSIGN FABS OVER 0< IF >R DNEGATE R>
        8000 OR THEN ;

: F+    ROT 2DUP >R >R FABS SWAP FABS -
        DUP IF DUP 0<
                 IF   ROT SWAP  NEGATE
                      R> R> SWAP >R >R
                 THEN 0 SWAP RALIGN
        THEN SWAP 0 R> R@ XOR 0<
        IF   R@ 0< IF 2SWAP THEN D-
             R> FSIGN ROT SWAP NORM
        ELSE D+ IF 1+ 2/ 8000 OR R> 1+
                 ELSE R> THEN THEN ;

: F-    FNEGATE F+ ;
: F<    F- 0< SWAP DROP ;
```

```
( FLOATING POINT INPUT/OUTPUT ) DECIMAL

CREATE PL 3 , HERE  ,001 , ,   ,010 , ,
          ,100 , ,           1,000 , ,
       10,000 , ,          100,000 , ,
    1,000,000 , ,       10,000,000 , ,
  100,000,000 , ,    1,000,000,000 , ,

: TENS  2* 2* LITERAL + 2@ ;      HEX
: PLACES PL ! ;
: SHIFTS FABS 4010 - DUP 0< NOT
        ABORT" TOO BIG" NEGATE ;
: F#    >R PL @ TENS DROP UM* R> SHIFTS
        RALIGN PL @ ?DUP IF 0 DO # LOOP
        ". HOLD THEN #S ROT SIGN ;
: TUCK  SWAP OVER ;
: F.    TUCK <# F# #> TYPE SPACE ;
: DFLOAT 4020 FSIGN NORM ;
: F     DFLOAT POINT TENS DFLOAT F/ ;
: FCONSTANT F 2CONSTANT ;

: FLOAT DUP 0< DFLOAT ;
: -+    DROP SWAP 0< IF NEGATE THEN ;
: FIX   TUCK 0 SWAP SHIFTS RALIGN -+ ;
: INT   TUCK 0 SWAP SHIFTS  ALIGN -+ ;

1.       FCONSTANT ONE DECIMAL
34.6680  FCONSTANT X1
-57828.  FCONSTANT X2
2001.18  FCONSTANT X3
1.4427   FCONSTANT X4

: EXP   2DUP INT DUP >R FLOAT F-
        F2* X2 2OVER FSQ X3 F+ F/
        2OVER F2/ F-      X1 F+ F/
        ONE F+ FSQ R> + ;
: FEXP  X4 F* EXP ;
: GET   BL WORD DUP 1+ C@ "- = TUCK -
        0 0 ROT CONVERT DROP -+ ;
: E     F GET >R R@ ABS 13301 4004 */MOD
        >R FLOAT 4004 FLOAT F/ EXP R> +
        R> 0< IF F/ ELSE F* THEN ;

: E.    TUCK FABS 16384 TUCK -
        4004 13301 */MOD >R
        FLOAT 4004 FLOAT F/ EXP F*
        2DUP ONE F<
        IF 10 FLOAT F* R> 1- >R THEN
        <# R@ ABS 0 #S R> SIGN 2DROP
        "E HOLD F# #>      TYPE SPACE ;
```

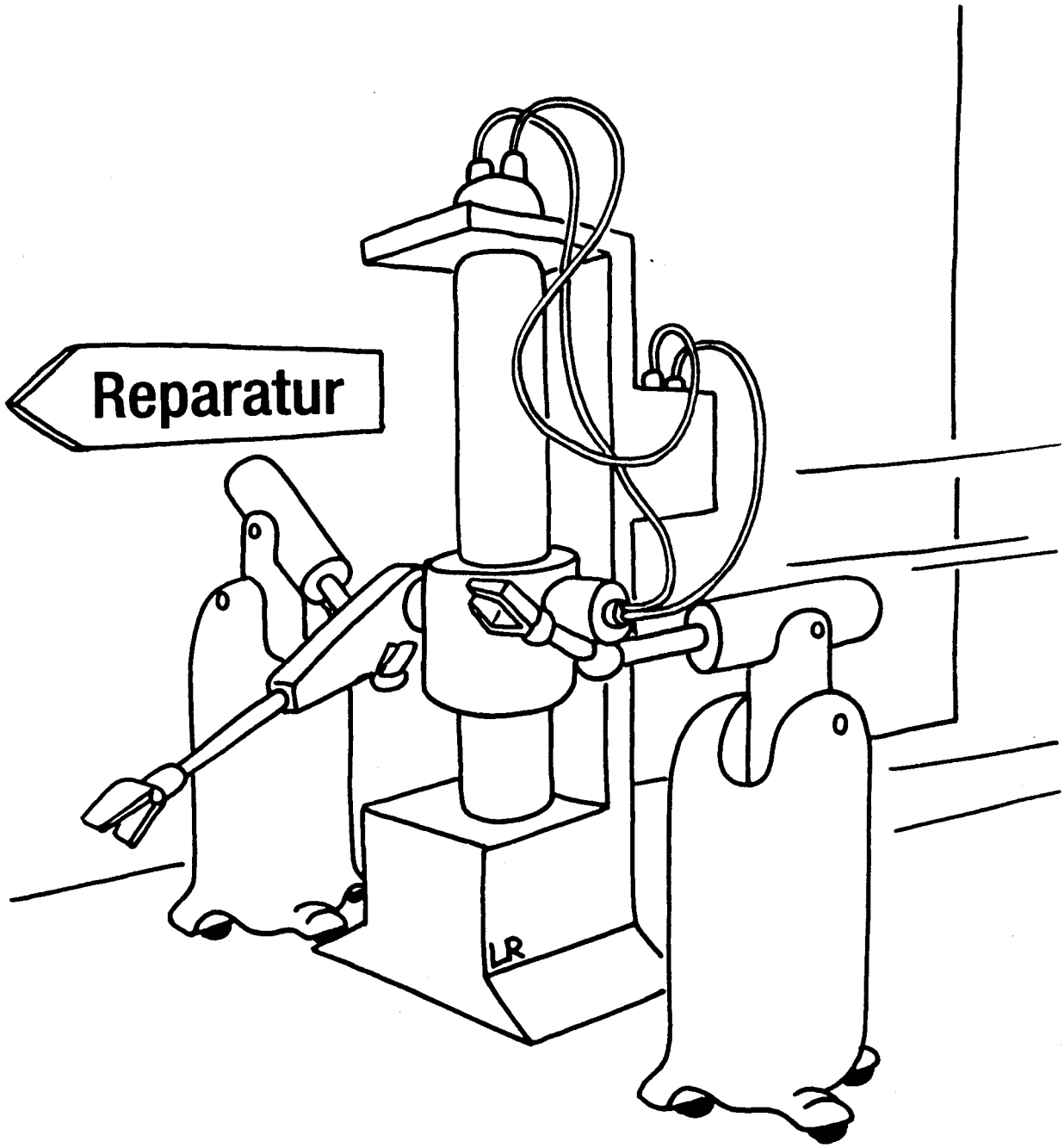
FORTH Dimensions

FEATURES

DEPARTMENTS



'Congratulations, Brian. You've just invented the world's first 2000 bit micro'



Nächstenhilfe

ALERT BOXEN

Bernd Pennemann, Hamburg

GEM-Programmierung in Forth

Eine Vorbemerkung

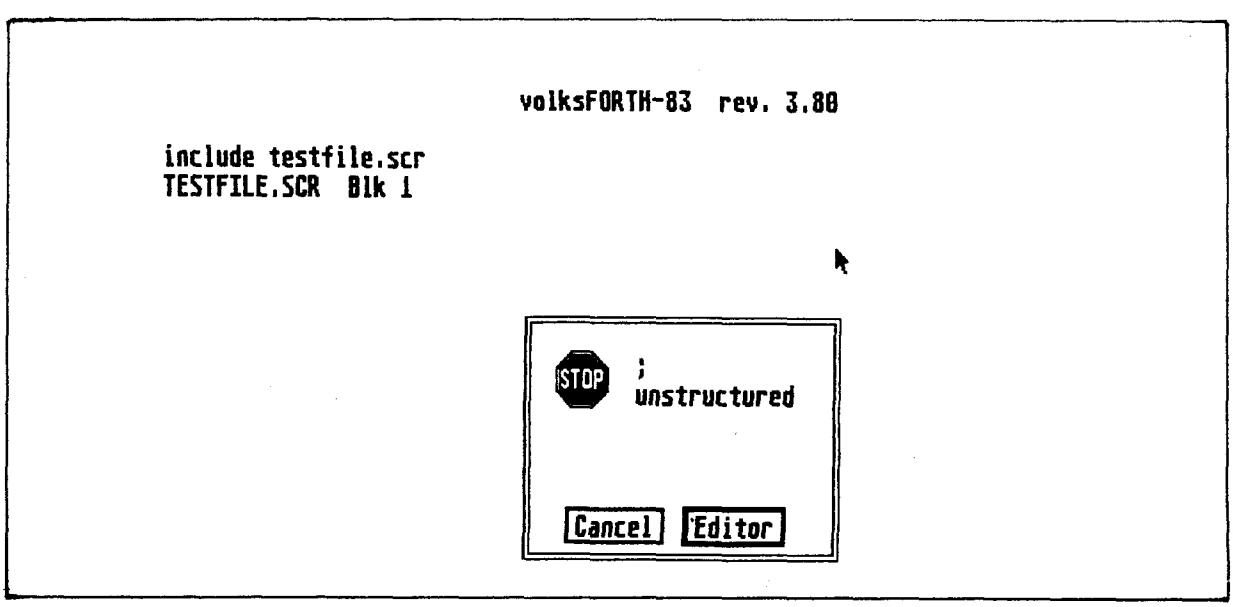
Die volksFORTH83-Versionen, die bisher im Umlauf waren, enthalten eine umfangreiche Bibliothek der GEM-Routinen. Diese Bibliothek gliedert sich in die Teile AES ("Application Environment System") und VDI ("Virtual Device Interface"). Im volksFORTH gehört auch ein Teil BASICS dazu, der die VDI und AES gemeinsamen Teile enthält.
In volksFORTH83 Versionen vor 3.80 funktionierten diese Bibliotheken nicht immer zufriedenstellend. Da wir alle volksFORTH83 umgetauscht haben, gehe ich daher davon aus, daß ein volksFORTH83 Version 3.80 zur Verfügung steht.

Bei meinen Arbeiten mit GEM mußte ich leider feststellen, daß GEM sehr absturzgefährdet ist. Ohne den Debugger, der im volksFORTH serienmäßig eingebaut ist, lassen sich die Absturzursachen kaum feststellen. Meine Erfahrungen mit "C" waren zumindest in dieser Hinsicht leidvoll. Daher darf man feststellen, daß Forth damit ein sehr geeignetes Werkzeug für die GEM-Programmierung darstellt.

Alert-Boxen unter volksFORTH83 auf dem Atari ST

Bernd Pennemann, Hamburg

In dem folgenden Artikel wird kurz skizziert, was unter "GEM" zu verstehen ist. Es werden sog. Alert-Boxen besprochen und schließlich wird ein Programm für das volksFORTH83 präsentiert, das dessen Fehlermeldungen dem Benutzer in diesen Boxen präsentiert. Ein Beispiel zeigt das folgende Bild :



Der Atari ST besitzt eine graphische Benutzeroberfläche, das sog. GEM (Graphics Environement Manager). Damit ist das Erstellen von sehr bedienungsfreundlichen Programmen möglich. Die Strukturen, die wir im GEM realisiert finden, sind das Ergebnis umfangreicher Untersuchungen bei AT&T, Murray Hill, NY. Sie sind in ähnlicher Art auf dem Macintosh realisiert und wurden kürzlich vorgestellt [1]. Zu GEM gehören Hilfsmittel zur Verwaltung von Windows, Menüs und sog. Objekten.
Windows benutzen Teile des Bildschirms, um Texte darzustellen. Typischerweise kann man die "Lage" dieses Windows auf dem darunterliegenden Text verändern, d.h. man kann sich unterschiedliche Teile des Textes ansehen. Ebenso kann man Lage und Größe des Fensters auf dem Bildschirm verändern.
Menüs gestatten das Auslösen von Aktivitäten eines Programms. Das besondere an den Menüs, wie man sie auf dem Atari und dem Macintosh findet, ist die Möglichkeit, sie jederzeit aufzurufen. Der Benutzer entscheidet, wann ein Menü aktiviert wird.
Objekte schließlich bestehen aus Textfeldern und Knöpfen. Knöpfe können durch Klicken mit der Maus ausgelöst werden, Texte können von der Tastatur aus eingegeben werden.
Das GEM besitzt nun bestimmte Objekte, sogenannte Alert-Boxen, um dem Benutzer auf einfache Weise nett aufgemachte Hinweise, Warnungen und Fehlermeldungen anzuzeigen. Alert-Boxen bestehen aus einem erklärenden Text, einem Hinweisschild, das keine tiefere Bedeutung besitzt und ein bis drei Knöpfen. Die Alert-Box ist Gegenstand dieses Artikels.
Das angegebene Programm ist insbesondere dann nützlich, wenn eigene Applikationen entwickelt werden sollen, die unter GEM laufen. Es gibt die Fehlermeldungen des volksFORTH in solchen Alert-Boxen aus.

Die ALERT-Box

Eine Alert-Box besteht aus drei Teilen. Zunächst ist da ein Pictogramm, das auch weggelassen werden kann. Es stehen drei Pictogramme zur Auswahl, nämlich "NOTE", "WAIT" und "STOP", die durch Nummern von eins bis drei ausgewählt werden (s.u.). Zwei davon können Sie auf den Illustrationen dieses Artikels erkennen.
Außerdem enthält die Alert-Box einen Text von maximal 5 Zeilen zu je 40 Zeichen, behauptet Atari. Schon bei mehr als 30 Zeichen gibt es jedoch Buchstabensalat.
Schließlich gibt es noch bis zu 3 Knöpfe, davon einen als Default-Knopf. Diese Knöpfe werden mit der Maus durch Anklicken ausgelöst, der Defaultknopf kann aber auch Drücken der Returntaste ausgelöst werden.
Die Routine, die solche Boxen malt, heißt FORM_ALERT. Ihr Aufruf sieht folgendermaßen aus :

```
FORM_ALERT      ( n1 adr -- n2 )
```

n1 ist die Nummer des Defaultknopfes, n2 die Nummer des gedrückten Knopfes. adr ist die Adresse eines mit $00 (Nullbyte) begrenzten Strings, der das Aussehen der Box bestimmt und folgendermaßen aufgebaut ist:

```
[ n ][ <Zeile1> <¦ Zeile2> .. ][ <Knopf1> < ¦ Knopf 2..]
```

Dabei ist n die Nummer des Pictogramms, Zeile1 der Text der ersten Zeile der Box und Knopf1 die Beschriftung des ersten Knopfes. Entsprechend für Zeile2, Knopf2 etc.

Eine Besprechung dieser Funktion findet man auch in der Literatur des Entwicklungspakets oder in [2].

Ein Beispiel finden Sie im folgenden Bild :

```
                    volksFORTH-83  rev. 3.80

gem also
: testbox       show_c  1  0" [1][Dies ist eine|Alert-Testbox][OK]"
                form_alert  drop ;
testbox
```

Beachten Sie bitte, daß der String mit einem Nullbyte abgeschlossen werden muß. Ich benutze dafür die Worte 0" ("null-string") und C>0" ("counted-to-null-string") , die in der GEM-Bibliothek des volksFORTH vorhanden sind. 0" legt, analog zu " einen String ab, jedoch nicht als counted string, sondern durch ein Nullbyte abgeschlossen. C>0" verwandelt einen counted string in einen durch ein Nullbyte begrenzten.

Die Maus

Damit Knöpfe gedrückt werden können, muß die Maus sichtbar gemacht werden (Man kann die Knöpfe tatsächlich auch nach Gehör auslösen, denn wenn man die Maustaste in der Alert-Box drückt, verschwindet der Pieps beim Klicken). Das geschieht durch SHOW_C . Beachten Sie bitte, daß die Maus nach dem Start des volksFORTH83 normalerweise abgeschaltet ist.

Noch ein paar Hinweise ..

In dem Beispielprogramm wird die Zeichenkette bei PAD mit Hilfe des Wortes $ADD zusammengebastelt. Der Quelltext enthält die Definition dieses Wortes, das aber im volksFORTH schon vorhanden ist.
Sollten Sie dieses Programm oder andere GEM-Routinen in eigenen Anwendungen verwenden, so dürfen Sie den Aufruf von GRINIT nach dem Starten Ihres Programms sowie GREXIT vor dem Verlassen des Forth nicht vergessen ! Andernfalls gibt es die herrlichsten

Abstürze. Das ist aber nur erforderlich, wenn der Editor in Ihrer Applikation nicht geladen ist; er führt nämlich diese Operationen durch, da er selbst das GEM benutzt. In diesem Fall sollten GRINIT und GREXIT nicht benutzt werden (sorry, ich kann auch nichts dafür) !
Bei eigenen Applikationen sollte man darauf achten, daß das Programm nach einem ABORT" ordentlich terminiert. In der Regel heißt das : An das unseres Wortes muß noch die Sequenz GREXIT BYE angefügt werden. Dadurch wird nach Malen der Box in das Betriebssystem zurückgesprungen.

Und nun viel Spaß beim GEMen !

[1] "Mach1, ein Forthsystem für den Macintosh", Holger Blum, Vierte Dimension, Vol II, Nr. 3 , pp 11.

[2] "Das große GEM-Buch zum Atari ST", Szcepanowski & Günther, Düsseldorf 1985, pp. 395

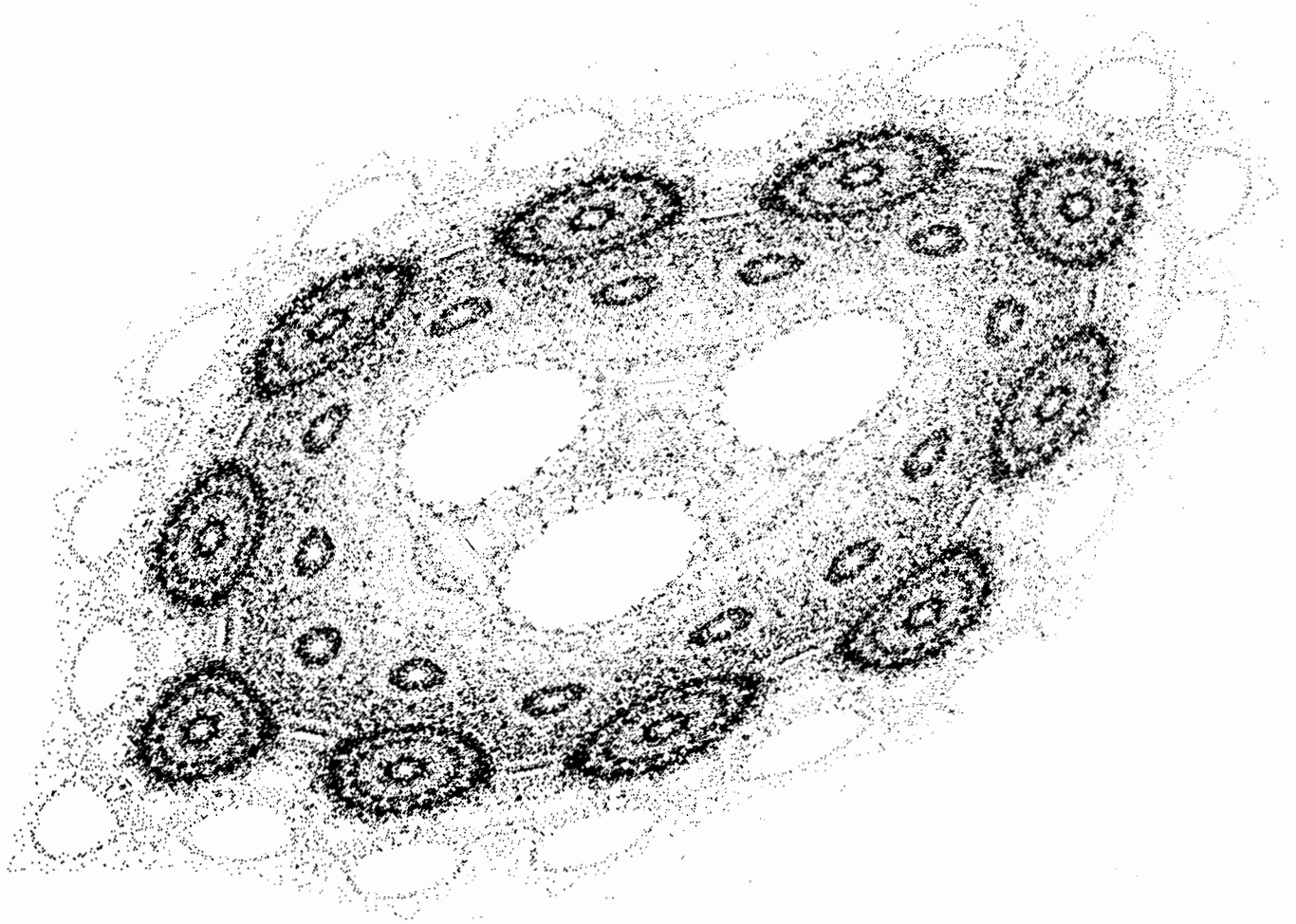
```
 volksFORTH-83     FORTH-Gesellschaft eV (c) 1985/86 we/bp/re/ks ERRORBOX.SCR  Seite 1

                        0                                        3

 0    ERRORBOX.SCR                                  bp 07nov86
 1
 2 Dieses File gibt alle ABORT"-Fehlermeldungen in ALERT-Boxen aus.
 3
 4 Diese Box enthält die Buttons "Cancel" und "Editor", falls der
 5    Fehler beim Laden eines Files auftrat. Der Button "Editor"
 6    verzweigt in den Editor, "Cancel" zum Kommandointerpreter.
 7    "Editor" ist der Defaultwert, der bei Drücken von (Return)
 8    ausgelöst wird.
 9 Trat der Fehler bei Ausführung von Tastatureingaben auf, gibt
10    es nur den OK-Button.
11
12
13
14
15

                        1                                        4

 0 \ Loadscreen for errorbox                     bp 07nov86 \ Loadscreen for errorbox                    26oct86we
 1
 2 \needs GEM                    include gem\aes.scr          Benötigt die AES-Bibliothek
 3
 4 Onlyforth Gem also definitions                             setzt Searchorder auf      GEM GEM FORTH ONLY    GEM
 5
 6 0 list         1 +load                                     Gebrauchsanleitung , kompiliert den folgenden Screen.
 7
 8 ' boxhandler errorhandler !                                setzt BOXHANDLER als neuen errorhandler. Alle Fehlermeldungen,
 9                                                             die Über abort" laufen, erscheinen jetzt in Boxen.
10 \ $ADD addiert einen String zu dem String, auf den $SUM zeigt.
11 \
12 \ Variable $sum                \ pointer to stringsum     $SUM          zeigt auf den String, an den $ADD hinten anbaut.
13 \
14 \ : $add      ( addr len -- )     dup )r                  $ADD          fügt einen String hinten an den String an, auf
15 \   $sum @ count +  swap  move   $sum @  dup c@  r) +  swap c! ;    den $SUM weist. Vorteil dieser Lösung : extrem einfach
```

2

```
0 \ Display all errors in an ALERT-Box                bp 07nov86
1
2 : addstring     ( string -- )        \ add a string to pad
3       count  $add ;
4
5 : boxhandler    ( string -- )
6    show_c       pad   dup off  $sum !
7    " [3][" addstring
8    here addstring
9    " |" addstring    addstring
10   blk @ ?dup IF    scr !   )in @ r# !
11                    2 " ][Cancel|Editor]"
12             ELSE  1 " ][Ok]"  THEN  addstring
13   pad c)0"   pad  form_alert  hide_c
14   2 = IF  v THEN  quit ;
15
```

5

```
\ Display all errors in an ALERT-Box                26oct86we

ADDSTRING  hängt den String bei Adresse string an den String
 bei $SUM an. Benutzt $ADD aus dem File STRINGS.SCR

BOXHANDLER   gibt den String von ABORT" in einer Alert-Box aus.
 Maus einschalten und PAD als Ziel für ADDSTRING vorbereiten.
 Die 3 sorgt für das STOP-Icon in der Box.
 Bei HERE steht das Wort, das den Fehler verursacht hat.
 In die nächste Zeile kommt die Fehlermeldung von ABORT"
 Wenn der Fehler beim Kompilieren auftrat, werden Screen und
   Cursorposition gemerkt und zwei Buttons ausgegeben.
 Sonst kann man den Fehler nur quittieren.
 Bei PAD ist jetzt der gesamte Boxtext zusammengestellt.
 Falls 'EDITOR' angeklickt wurde, wird der Editor mit dem
 fehlerhaften Screen aufgerufen.
```



MASCHINENSCHREIBEN

Michael Kalus, Schwelm

Flüssiges Maschinenschreiben erfordert systematisches Üben auf der Tastatur. Zur Grifferarbeitung der Tastatur nach DIN 2137 haben Behrens und Ranft einen Lehrgang erarbeitet (1), der mir gut gefallen hat. Die Übungen habe ich in Screens aufgenommen, das Programm dazu stellt die gewünschten Übungen vor, und man kann dann Zeile für Zeile mit steigendem Schwierigkeitsgrad den Fingersatz üben.

Geübt wird der Anschlag 'Zehn Finger blind' ausgehend von der Grundstellung der Schreibfinger der linken Hand auf den Tasten a s d f und der rechten Hand auf j k l ö. Drückt man eine falsche Taste, wird dies sofort mit einem BEEP quitiert. Das falsche Zeichen kann mit DEL korrigiert werden.

Was Sie aus dem File UEBUNGS.TXT abgeschrieben haben, wird im File TEST.TXT aufgehoben für den Fall, daß eine Bewertung der Schreibleistung durchgeführt werden soll. Wenn Sie eine Uhr in Ihrem Rechner-System haben, kann das automatisiert werden. In meinem alten AppleII gibt es (noch) keine Uhr, und daher habe ich keine Bewertung implementiert. Die Kalkulationen dafür sind aber simpel, die Anleitung dazu finden Sie in Bild 1.

Es war purer Zufall, daß die Übungen sich so wunderschön in Forth-Screens darstellen liessen. Zudem stellt Laxen & Perry's Forth-83-Standard-Model, das F83, bereits so viele Utilities bereit, um mit Files und Buffern umzugehen, daß nur noch wenig Forthcode nötig war, um den Tippkurs T.COM daraus zu erzeugen. Dabei ist die ANWENDER HILFE und ein GLOSSAR der Worte für den Anwender bereits mitgerechnet.

Natürlich war der Kurs im Maschinenschreiben in erster Linie für mich als Forth-Benutzer gedacht. Doch er war auch für ANWENDER ohne Kenntniss von Computern leicht begreiflich. Schwierigkeiten gab es eigentlich nur mit dem Weg durch das Betriebssystem CP/M bis zu der Selle, wo T.COM aufgerufen wird. Ab da kamen meine 'Testpiloten' gut selbst zurecht.

In Screens 2-6 finden Sie den Forthcode und in den Shadow-Screens 11-13 die Kommentare dazu. In der Applikation gibt es zahlreiche Stellen, die nicht streng zum 83-Standard, sondern zu den Utilities von Laxen & Perry gehören. Es würde hier zu weit führen, diese ebenfalls zu besprechen. Aber dem näher interessierten Leser stehe ich gerne mit Rat zur Seite und schicke auf Wunsch das Source-Listing UTILITY.BLK des F83 zu.

Die Bewertung der Schreibfertigkeit

Mindestanforderungen (nach dem Lehrplan an bayerischen[1] Schulen):

bei 2 Wochenstunden = 80 Anschläge/Min.
bei 3 Wochenstunden = 100 Anschläge/Min.

Die Bewertungsgrundlage

Der Wert einer Arbeit wird beurteilt anhand der Relation Fehler : Schreibgeschwindigkeit.

Note 1 bis einschließlich 0,125 % Fehlern,
Note 2 bei 0,126 % bis 0,250 % Fehlern,
Note 3 bei 0,251 % bis 0,400 % Fehlern,
Note 4 bei 0,401 % bis 0,550 % Fehlern,
Note 5 bei 0,551 % bis 0,700 % Fehlern,
Note 6 bei mehr als 0,700 % Fehlern.

Die Fehlerprozente werden wie folgt ermittelt:

$$\frac{\text{Zahl der Fehler} \times 100}{\text{Gesamtanschläge}}$$

z. B.: $\frac{2 \times 100}{1030} = \frac{200}{1030} = 0{,}194\ \%$

Die Herabstufung bei Minderleistungen

Die Note wird herabgesetzt
1. um eine Stufe bei 1 bis 10 Anschlägen/Min. unter der Mindestanschlagszahl,
2. um zwei Stufen bei 11 bis 20 Anschlägen/Min. unter der Mindestanschlagszahl.

Die Note 6 wird außerdem erteilt, wenn die Anschlagszahl
1. um mehr als 20 Anschläge/Min. unter der Mindestanschlagszahl liegt,
2. unter der Mindestanschlagszahl liegt und die Zahl der Fehler mehr als vier beträgt.

[1] An den Schulen anderer Bundesländer gelten für die Bewertung der Schreibfertigkeit die jeweils erlassenen Prüfungsordnungen.

```
      1
 0 \ Load screen  "Maschinenschreiben"                07jul86mk
 1 decimal
 2
 3 2 8 thru
 4
 5 \ ' hallo is boot
 6 \   save-system T.COM
 7
 8
 9
10
11
12
13
14
15
```

```
      2
 0 \ variables, constants and stuff                   15jun86mk
 1 only forth also definitions
 2
 3 variable 'INB      : INB  (S -- adr)  'inb @ ;
 4 variable 'EXB      : EXB  (S -- adr)  'exb @ ;
 5 variable 'ROW      : ROW  (S -- n )   'row @ ;
 6 variable 'HALT     : HALT?  (S -- f )  'halt @ ;
 7
 8         0 constant TAB1
 9   tab1 4 + constant TAB2
10 tab2 c/l + constant TAB3
11
12 : SAVE-ROW  (S -- )  #line @ 'row ! ;
13 : POS1  (S -- )  tab1 row at ;
14 : POS2  (S -- )  tab2 row 1+ at ;
15
```

```
      3
 0 \ (exercise)                                       07jul86mk
 1 : !BUFFERS  (S n -- )
 2    c/l * scr @ 2dup block + 'inb ! in-block + 'exb ! ;
 3 : LETTER  (S n a -- c )   + c@ ;
 4 : INCORRECT?  (S n+1 -- f)
 5    1- dup  exb letter  swap  inb letter  <> ;
 6 : (COMPARE-CHAR)  (S a n char -- a n+1)
 7    (char) dup incorrect?  if beep then ;
 8 : (EXERCISE)  (S n -- )
 9    dup !buffers  pos1 3 .r space  exb c/l type
10    pos2  c/l spaces   pos2  inb c/l expect
11    pos1  c/l 4 + spaces ;
12
13
14
15
```

```
      11
Hinweis:                                              08jul86mk

Der Uebungstext ist urheberrechtlich geschuetzt und nicht
Teil dieser Public Domain Software. Er muss selbststaendig
im File UEBUNGS.TXT angelegt werde.
```

```
      12
                                                      15jun86mk

INB  Input Buffer Adresse fuer die eingetippte Zeile.
EXB  Example Buffer Adresse fuer die vorgestellte Uebungszeile.
ROW  Nummer der gesicherten Zeile.
HALT?  Liefert das Flag mit dem die Schleife EXERCISE verlassen
  werden kann. Wird mit  <Ctrl-Q>  gesetzt.
TAB1  Tabulatorposition
TAB2  Tabulatorposition
TAB3  Tabulatorposition

SAVE-ROW  Sichert die Zeilennummer fuer POSn .
POS1  Oefter wiederkehrende Position.
POS2  Oefter wiederkehrende Position.
```

```
      13
                                                      07jul86mk
!BUFFERS  Berechnet INB und EXB aus der Zeilennummer n.

LETTER  holt ein Zeichen von der Adresse a+n.
INCORRECT?  Testet die Zeichen der Position n+1
  in INB und EXB auf Ungleichheit.
(COMPARE-CHAR)  Bildet das eingegebene Zeichen ab, vergleicht
  es mit dem Zeichen der Uebung und quitiert richtig/falsch.
(EXERCISE)  Bietet die Zeile  i  zum ueben an. Die aktuelle
  Uebung ist in Screen #  n  im Uebungstext-File.
```

Fortsetzung auf Seite 32

Self-Understanding Programs

Mitch Bradley

ABSTRACT

This presentation describes a wordset that makes it possible to write portable programs which understand themselves. The most obvious example is a Forth decompiler, which is traditionally non-portable.

The Problem

It is not possible to write a "standard" decompiler, because the details of what goes into memory when compiling a colon definition are left to the implementor. A decompiler must know these details, thus it is non-portable. Other interesting programs face the same problem (e.g. Kim Harris's Structure Chart program[1]).

Such programs are examples of "Self-Understanding Programs", programs that know the format in which they are stored. Self-Understanding is possible in a few other languages, such as Lisp and PostScript. In these languages, a program is a visible data type. In Lisp, a program is just a list, which is the basic Lisp data type. In PostScript, which is very much like Forth in many ways, the equivalent of a colon definition is compiled as an array of objects, and a PostScript "word" is one kind of object.

In constrast, there is no standard way in which the body of a Forth colon definition is stored in memory.

The Solution

Rather than constrain Forth implementations by specifying details of how definitions are compiled, it is sufficient to define a set of words which can "unravel" the compiled definitions.

In the process of porting a decompiler to four different Forth implementations, I have developed a relatively small set of Forth words which are helpful in encapsulating the implementation details of the system. The key observation is that a colon definition body contains only a few different kinds of things: references to other Forth words, branches, literals, and string literals (like ."). The new wordset understands the way that references, branches, literals and string literals are compiled. Only this small set of words has to be re-implemented for a new system, and the decompiler is largely portable.

This wordset can be used to write other programs, not just decompilers, which can decode and understand compiled Forth programs.

The wordset works well for different threading schemes (direct, indirect, token, subroutine), different branch styles (absolute, relative), and different address lengths (16-bit, 32-bit, absolute, relative). It is probably insufficient to deal with Forth systems that compile directly to native code [2], but it may be helpful even there.

The Colon Definition Model

This scheme assumes that Forth colon definitions contain the following kinds of things:

Tokens

A Token is a compiled reference to another Forth word. This does not imply the use of token-threaded code. Tokens can be absolute addresses, relative addresses, indices into a

table, jump-to-subroutine instructions, or whatever the system compiles when a Forth word is referenced inside another. Another way of looking at the same thing is that a token is the way that a compilation address is stored within the body of a colon definition.

Branches

A Branch is something that is compiled inside a Forth definition in order to change the order of execution of Tokens. Each Branch has a destination, which is the the new value of the Interpreter Pointer if the branch is taken. Different implementations compile the branch destination in different ways; absolute addresses, relative addresses, etc. Examples of branches are the run-time words compiled by IF , ELSE , WHILE , REPEAT , LOOP , etc.

Literals

A Literal is a parameter that is compiled "in-line" inside a colon definition. The most common example is a compiled number.

String Literals

A String Literal is a string that is compiled into a colon definition, perhaps by ." or ABORT" . The implementation differences between compiled strings usually result from alignment restrictions of various processors. For example, the 68000 requires a 16-bit item to be stored at an even address, so compiled strings are usually padded so that they end on an even boundary.

Properties of Words

Each word, regardless of its type (colon definition, code definition, variable, constant, etc) is assumed to have these attributes:

Compilation Address

The compilation address of a word is the address which is returned by FIND. If the word is headerless, it still has a compilation address, which is the address that would have been returned by FIND before the name was thrown away.

Name

The name is some string which represents the word. The "real" name of the word cannot always be determined. For instance, the word may be headerless or the name stored in the name field may be truncated, as in systems that only store the first 3 characters of the name. In the first case, the name could be represented as Wnnnn, where nnnn is the hex representation of the compilation address of the word. In the second case, the name could be represented as, for example, INT------, where the dashes show the missing characters.

Immediate Flag

A word is either IMMEDIATE or not. If the word is headerless, in which case it may be impossible to determine whether or not the word is immediate, it is reasonable to assume that the word is not immediate, since headerless compiling words aren't used often.

Definer

The Definer of a word "A" is the compilation address of the word "B" which created the word "A". For a colon definition, the definer is the compilation address of the word ":".

The Wordset

The *specification* and use of the following words is system independent and portable. The *implementation* is highly system dependent, hence I do not show the implementation.

A unifying rule followed here is that an address is always represented *on the stack* as an absolute address, regardless of how it is stored inside a colon definition. This is true for both compilation addresses and branch destinations. If the address is on the stack, it is absolute.

TOKEN@ (addr -- cfa)

cfa is the (absolute) compilation address which is compiled at addr. In a system that compiles absolute addresses, TOKEN@ is the same as @ . Otherwise, TOKEN@ has to do whatever is necessary to put an absolute compilation address on the stack.

TOKEN! (cfa addr --)
The absolute compilation address cfa is stored at addr. In a system that compiles absolute addresses, TOKEN! is the same as ! . Otherwise, TOKEN! compiles cfa in some system-dependent manner.

/TOKEN (-- n)
Pronounced *per token.* n is the length in bytes of a compiled token. For example, a system which compiles 16-bit absolute addresses would have /TOKEN equal to 2.

>TARGET (addr -- destination-addr)
Pronounced *to target.* destination-addr is the branch destination address of the (high level) branch that is compiled at addr. Note that addr is the address of the branch instruction, not the offset word. For instance, if the system compiles BRANCH -10 for REPEAT , addr would be the address of the BRANCH .

/BRANCH (-- n)
Pronounced *per branch.* n is the length in bytes of compiled branch destination. For example, a system which compiles 1-byte relative branch offsets would have /BRANCH equal to 1. A system which compiles 16-bit branch offsets would have /BRANCH equal to 2.

+STR (addr1 -- addr2)
addr2 is the first available address after the string compiled at addr1, taking into account any padding that may be appended to the string.

>DATA (cfa -- data-address)
Pronounced *to data.* data-address is the address of the data storage area associated with the word whose compilation address is cfa . For example, for variables, >DATA is equivalent to >BODY , for user variables, >DATA returns an address in the user area, etc. The implementation of this is not necessarily very efficient.

DOES-IP? (addr1 -- addr2 flag)
addr1 is assumed to point just after the run-time word compiled by ;CODE or DOES> . flag is true if the code at addr is a DOES> clause as opposed to a ;CODE clause. In the case of a ;CODE clause, addr2 points to the actual machine code that is the ;CODE clause, skipping whatever linkage code was necessary to set up the execution of the clause. In the case of a DOES> clause, addr2 points to the first token that was compiled as part of the DOES> clause, skipping whatever linkage code was necessary to set up the execution of the clause. This is a necessary kludge.

C.ID (cfa --)
Displays the name of the word whose compilation address is cfa. This is similar to the phrase >NAME .ID , but it must cope properly with the case of headerless code. If the word is headerless, C.ID displays the string Wnnnn, where nnnn is the hex representation of the compilation address cfa .

IMMEDIATE? (cfa -- flag)
Flag is true if the word whose compilation address is cfa is immediate. If the correct answer cannot be determined, as for instance in a headerless definition, flag is false (which may be wrong, but is nevertheless a reasonable guess).

/N (-- n)
n is the number of bytes occupied by a number that is stored in memory. This is necessary for portability between 16 and 32 bit implementations. [3]

FOLLOW (voc --)
Used for scanning the list of words in a vocabulary. Initializes a system-dependent data structure in preparation for scanning a vocabulary. voc-threads is a system-dependent number which represents a vocabulary. It is the same number that is stored in CONTEXT and CURRENT . FOLLOW is used in conjunction with ANOTHER? .

ANOTHER? (-- nfa true OR false)
If there is another word in the vocabulary currently being scanned, nfa is the name field

address of that word and the flag is true. If there are no more words, the flag is false. Used in conjunction with FOLLOW . Example: To print the names of all the words in the CURRENT vocabulary,

```
: WORDS ( -- )
  CONTEXT @ FOLLOW
  BEGIN  ANOTHER?  WHILE  .ID  REPEAT
;
```

>THREADS (cfa -- voc-threads)

cfa is the compilation address of a vocabulary. voc-threads is the number which is stored in CONTEXT when that vocabulary is executed. I.e. " ' FORTH >THREADS " and " FORTH CONTEXT @ " both leave the same number on the stack (but the second form has the side effect of also affecting the search order).

Other Useful Portability Words

These words do not fit into the category of building programs which understand Forth definitions. However, I have found them useful in writing portable Forth code for other purposes.

ALIGNED (addr1 -- addr2)

Addr2 is the result of aligning addr1 to the "natural" boundary for the particualar system. Addr2 is always greater than or equal to addr1. For example, in most 68000 implementations ALIGNED would move up to the next even boundary. In most 8 bit processor implementations, ALIGNED would be a no-op.

ALIGN (--)

Advance the dictionary pointer to the next alignment boundary. This can be defined in a portable fashion in terms of ALIGNED .

SKIPSTR (-- addr len)

This is a very important word. It is compiled as part of the definition of a run-time word which is followed by a string literal. When SKIPSTR executes, it leaves the address and length of its string literal, and advances the Interpreter Pointer past the string. For example, the run time word for ." could be defined as:

```
: (.") ( -- )  SKIPSTR TYPE ;
```

SKIPSTR could be defined in some systems with:

```
: SKIPSTR ( -- addr len )
  R> COUNT  2DUP +  ALIGNED  >R
;
```

The use of SKIPSTR replaces many instances of phrases like " R> COUNT + >R ", which are not always portable because of differences in the way that systems compile in-line strings. SKIPSTR is a good candidate for a CODE word.

FIND-CFA (addr -- cfa)

addr is an address within the body of a forth word. cfa is the compilation address of that word. This word is difficult to implement so that it is guaranteed to work, but is fairly easy to implement so that it works most of the time. It is useful for things like a word to unravel the return stack and print the names of the words on it. It is also useful for implementing the word DEFINER.

Discussion

Using these words, I have moved a number of utilities, previously considered highly system-dependent, across several Forth implementations with widely differing implementation details. The wordset does not ensure complete portability, but it helps a lot. The problems that remain are mostly the different names that different systems give to the run-time words compiled

by branches and various kinds of literals. These problems are easily handled by a table which enumerates the various run-time words. The wordset described here takes care of the annoying details which otherwise tend to get buried in hard-to-find places in the code.

Acknowledgements

The FOLLOW / ANOTHER? scheme was invented by Gordon Smith. My portable decompiler started with the decompiler in the Laxen/Perry F83 model [4], and was then extended to do structured decomilation of compiling words, to do sensible things with children of user-defined compiling words, and to be portable. Mike Perry suggested the word >DATA .

References

1. Sebok, William L. "A VAX Immplementation of the FORTH-79 Standard", *The Journal of Forth Application and Research*, Vol. 2 No. 4, 1984.
2. Harris, Kim, "Structure Charts", presented at the Silicon Valley Chapter FIG Meeting, October 1985.
3. Bradley, Mitch, and Sebok, William L. "Compatible Forth on a 32-bit Machine", *The Journal of Forth Application and Research*, Vol. 2 No. 4, 1984.
4. Perry, Michael, "F83, a Public Domain Model Implementation of the Forth-83 Standard", *Proc. 1983 Asilomar FORML Conference".*

Weitere Literaturhinweise in diesem Heft

FORTH GRUPPEN

Hamburg
  Treffen jeden vierten Samstag im Monat ab 16:00Uhr in der Berufsfachschule für Radio- und Fernsehtechnik, Eimsbüttelerstr.64-66.
  Kontakt: Bernd Pennemann, 040-6900539
Karsruhe
  Treffen jeden dritten Mittwoch im Monat ab 19:00Uhr im Jugend- und Begegnungszentrum, Krohnenplatz.
  Kontakt: Michael Weiss, 0721-854994
Paderborn
  Treffen in der Gruningerstr.20 nach Verabredung.
  Kontakt: Thomas Asche, 05251-26496
Wuppertal
  Treffen jeden vierten Freitag im Monat ab 20:00Uhr im Bahnhof Ottenbruch, Funckstrasse, W'tal-Elberfeld.
  Kontakt: Michael Kalus, Präsidentenstr.40, 5830 Schwelm, 02336-82204

volksFORTH-83
  Kontakt: Bernd Pennemann, 040-6900539

Graphik und Animation
  Kontakt: Marco Pauck, 040-3900139

Berlin (*)
  Kontakt: Hans Madlung, 030-4141831
Braunschweiger Raum (*)
  Kontakt: Eckhard Heyne, 05352-58087
Darmstadt (*)
  Kursus in der Volkshochschule. Kontakt: Andreas Soeder, 06257-2744
Freiburger Raum (*)
  Kontakt: Markus Gimbel, 07641 - 42819
Moers (*)
  Kontakt: Hans Chrapia, 0203-3793274
Münchner Raum (*)
  Kontakt-1: Heinz Schnitter, privat 089-3103385, Labor 089-32095276.
  Kontakt-2: Jürgen Helbig und Robert Schörghuber, 089-2283531
  Kontakt-2: Ekkehard Flögel, 08021-8414
Rhein/Nekar Raum (*)
  Thomas Prinz, priv 06271-2830

Villach Östereich (*)
  Forth-Club in Kärnten. Treffen nach Verabredung. Kontakt: Heinz Klambauer, 04242-33566

Belgien FIG Chapter
  Kontakt: Luk van Loock, Lariksdreff 20, 2120 Schoten, Telefon 03-658-6343

Holland FIG Chapter
  Kontakt: Adriaan van Roosmalen, Heusden Houtsestraat 134, 4817 We Breda, Telefon 31-76-713104

Schweiz FIG Chapter
  Kontakt: Max Hugelshofer, ERNI & Co., Elektro-Industrie, Stationsstrasse, 8306 Bruttisellen,
  Telefon 01-833-3333

(*) = Forth-Enthusiasten oder Gruppen, die noch nicht offiziell als Gruppe der FG anerkannt sind bzw. zu dem Zweck noch Verbindungen zu weiteren Forthlern suchen. Wenn Sie interessiert sind, kann hier auch ihr Name stehen. Schreiben Sie an die VIERTE DIMENSION.

Maschienenschreiben

```
      4                                                    07jul86mk
 0 \ adapt system
 1 : HALT  (S m a n c -- m a n )  'halt on  cr-in ;
 2
 3 create CC-TYPING
 4 ] drop  drop  drop  drop  drop  drop  drop  drop
 5   drop  drop  drop  drop  drop  CR-IN drop  drop
 6   drop  HALT  drop  drop  drop  drop  drop  drop
 7   drop  drop  drop  drop  drop  drop  drop  drop  [
 8
 9 : ADAPT  (S -- )
10   ['] (compare-char) is char    cc-typing cc ! ;
11 : READAPT  (S -- )
12   ['] (char) is char     cc-forth cc ! ;
13
14
15
```

```
      5                                                    08jul86mk
 0 \ define  files
 1 dos also definitions
 2 : IN-DEFINE  (S -- )
 3   define  Does> in-file ! ;
 4 in-define UEBUNGS.TXT    define TEST.TXT
 5
 6 forth definitions
 7 : IN-LIST  (S -- )  dark switch list switch cr cr ;
 8 : LESSON  (S -- )
 9   test.txt open-file   uebungs.txt open-file ;
10 : EXERCISE  (S n -- )
11   in-list  cr cr save-row   adapt  'halt off
12   16 1 do  i (exercise)  halt? if leave then  loop
13   readapt   pos2 cr ;
14 : GLOSSARY  (S -- )
15   0 in-list ;
```

```
      6                                                    07jul86mk
 0 \ german overlay
 1 vocabulary ANWENDER    Anwender definitions
 2 : BYE  (S -- )  bye ;
 3 : LISTEN  (S n -- )  in-list ;
 4 : UEBEN  (S n -- )  exercise ;
 5 : GLOSSAR  (S -- )  glossary ;
 6 : HILFE  (S -- )
 7   only forth definitions  also Anwender words  lesson ;
 8
 9 only forth also Anwender also forth definitions
10 : HALLO  (S -- )
11   cr ." Kursus im Maschinenschreiben"
12   cr ." Version 2/1"
13   cr cr
14   cr ." Programmiert mit dem"
15   hello  cr cr  hilfe ;
```

```
      14                                                   07jul86mk
HALT  Setzt das Flag mit dem die Uebung angehalten wird.

CC-TYPING  Dies ist eine Control-Code Tabelle fuer EXPECT.
  <Ctrl-Q> = HALT
  <Ctrl-M> = <RETURN> = Ende der Zeicheneingabe
   uebrige = Keine Aktion


ADAPT  Bereitet EXPECT fuer die Anwendung vor.

READAPT  Setzt EXPECT zurueck passend fuer F83.
```

```
      15                                                   07jul86mk
IN-DEFINE Definiert ein File aus dem gelesen wird.
TEST.TXT  Die Uebung wird in dieses File eingetippt.
UEBUNGS.TXT  Der Uebungstext steht in diesem File.


IN-LIST  listet die Screens des 'in-file'.
LESSON  Oeffnet die Files TEST.TXT und LESSON.TXT fuer die
  Uebung.
EXERCISE  Fuehrt die Uebung n aus. Ein Screen entspricht einer
  Uebung. Geuebt wird Zeilenweise. Es gelten die Control Codes
  der Tabelle CC-TYPING.
GLOSSARY  erklaert die Anwenderworte.
```

```
      16                                                   07jul86mk
ANWENDER  Vocabular der Anwenderworte.
BYE  Zurueck ins Betriebs-System.
LISTEN  Zum ansehen der Uebungen.
UEBEN  Zur Grifferarbeitung muss man fleissig  n  UEBEN .
GLOSSAR  Siehe GLOSSARY.
HILFE  Fuer der Fall, dass ein Anwender seine Worte vergessen
  hat. Und wenn Sie im Forth waren, sagen Sie ANWENDER HILFE
  und die Applikation ist wieder da.

HALLO  Erscheint beim booten der Anwendung.
```