



Das Forth-Magazin

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

In dieser Ausgabe:



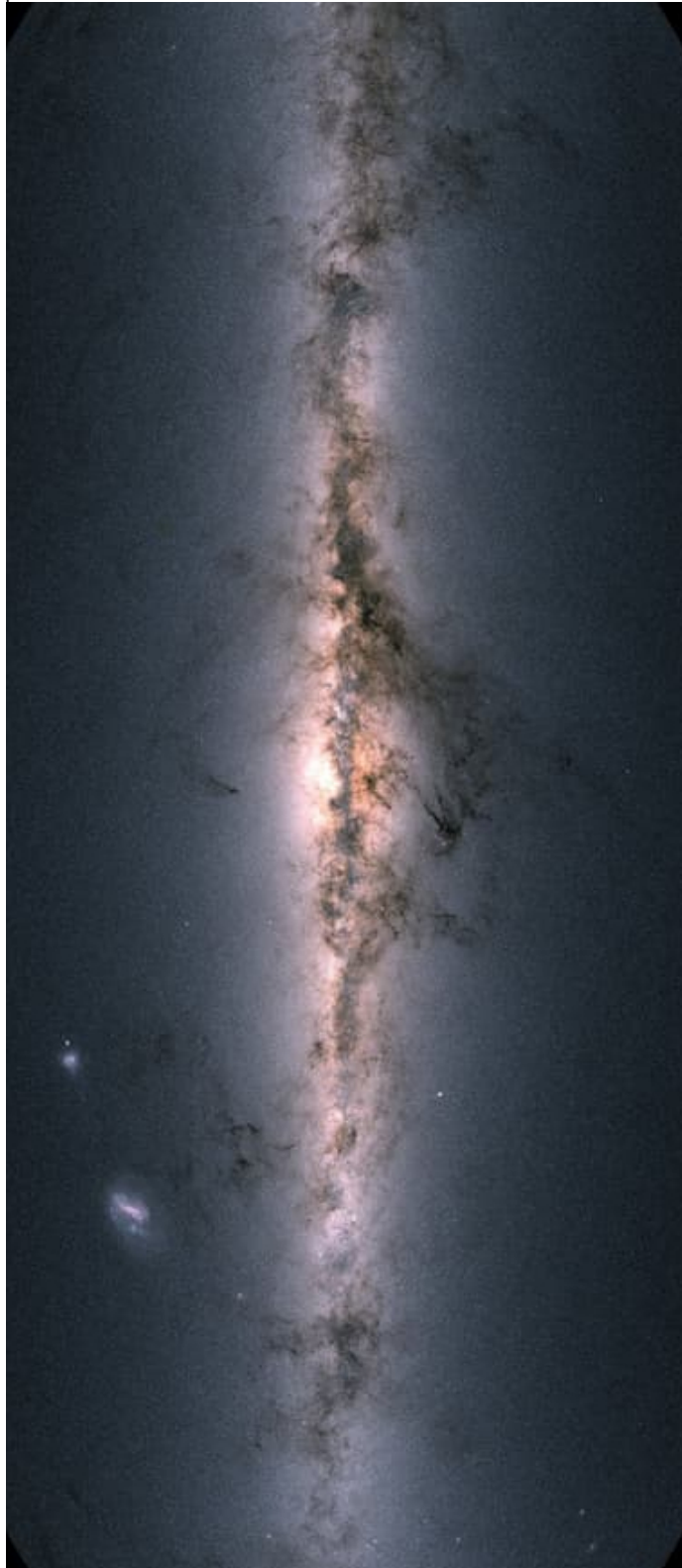
Astroimaging in FORTH: Data
Formats and Computations

Statische Typprüfung in Forth

Locals in Fancy-Forth

Z79 Forth Reference Board V0.3

Mein „Starting Forth“



Servonaut



Fahrtregler - Lichtanlagen - Soundmodule - Modellfunk

tematik GmbH
Technische
Informatik

Feldstraße 143
D-22880 Wedel
Fon 04103 - 808989 - 0
Fax 04103 - 808989 - 9
mail@tematik.de
<http://www.tematik.de>

Seit 2001 entwickeln und vertreiben wir unter dem Markennamen „Servonaut“ Baugruppen für den Funktionsmodellbau wie Fahrtregler, Lichtanlagen, Soundmodule und Funkmodule. Unsere Module werden vorwiegend in LKW-Modellen im Maßstab 1:14 bzw. 1:16 eingesetzt, aber auch in Baumaschinen wie Baggern, Radladern etc. Wir entwickeln mit eigenen Werkzeugen in Forth für die Freescale-Prozessoren 68HC08, S08, Coldfire sowie Atmel AVR.

Forth-Schulungen

Möchten Sie die Programmiersprache Forth erlernen oder sich in den neuen Forth-Entwicklungen weiterbilden? Haben Sie Produkte auf Basis von Forth und möchten Mitarbeiter in der Wartung und Weiterentwicklung dieser Produkte schulen?

Wir bieten Schulungen in Legacy-Forth-Systemen (FIG-Forth, Forth83), ANSI-Forth und nach den neusten Forth-200x-Standards. Unsere Trainer haben über 20 Jahre Erfahrung mit Forth-Programmierung auf Embedded-Systemen (ARM, MSP430, Atmel AVR, M68K, 6502, Z80 uvm.) und auf PC-Systemen (Linux, BSD, macOS und Windows).

Carsten Strotmann carsten@strotmann.de
<https://forth-schulung.de>

RetroForth

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

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



Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

Weitstraße 140
80995 München
sales@cornu.de
www.cornu.de

Unser Themenschwerpunkt ist automotive SW unter AutoSAR. In Forth bieten wir u. a. Lösungen zur Verarbeitung großer Datenmengen, Modultests und modellgetriebene SW, z. B. auf Basis eCore/EMF.

KIMA Echtzeitsysteme GmbH

Güstener Straße 72 52428 Jülich
Tel.: 02463/9967-0 Fax: 02463/9967-99
www.kimaE.de info@kimaE.de

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

FORTEch Software GmbH

Tannenweg 22 m D-18059 Rostock
<https://www.fortech.de/>

Wir entwickeln seit fast 20 Jahren kundenspezifische Software für industrielle Anwendungen. In dieser Zeit entstanden in Zusammenarbeit mit Kunden und Partnern Lösungen für verschiedenste Branchen, vor allem für die chemische Industrie, die Automobilindustrie und die Medizintechnik.

Ingenieurbüro Tel.: (0 82 66)–36 09 862
Klaus Kohl-Schöpe Prof.-Hamp-Str. 5
D-87745 Eppishausen

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

Mikrocontroller-Verleih Forth-Gesellschaft e. V.

Wir stellen hochwertige Evaluation-Boards, auch FPGA, samt Forth-Systemen zur Verfügung: Cypress, RISC-V, TI, MicroCore, GA144, SeaForth, MiniMuck, Zilog, 68HC11, ATMEL, Motorola, Hitachi, Renesas, Lego ...
<https://wiki.forth-ev.de/doku.php/mcv:mcv2>

Leserbriefe und Meldungen	5
Astroimaging in FORTH: Data Formats and Computations	8
<i>Andrew Read</i>	
Statische Typprüfung in Forth	12
<i>Stephan Becher</i>	
Locals in Fancy-Forth	18
<i>Jörg Völker</i>	
Z79 Forth Reference Board V0.3	23
<i>Rafael Deliano</i>	
Mein „Starting Forth“	25
<i>Johannes Reilhofer</i>	

Titelbild: **Karte der Milchstraße** Der ESA-Satellit Gaia scannt seit 2014 die Milchstraße ab. Diese Karte unserer Heimatgalaxie enthält 1,7 Milliarden Sterne, deren Helligkeit und Position Gaia zwischen Juli 2014 und Mai 2016 vermessen hat.

Quelle: Aus dem Internet gefischt, modifiziert. mk

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth-Gesellschaft e. V.

Postfach 1030

48481 Neuenkirchen

E-Mail: Secretary@forth-ev.de

Direktorium@forth-ev.de

Bankverbindung: Postbank Hamburg

BLZ 200 100 20

Kto 563 211 208

IBAN: DE60 2001 0020 0563 2112 08

BIC: PBNKDEFF

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann

E-Mail: 4d@forth-ev.de

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Manuskripte und Rechte

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

Liebe Leser,

schon gesehen, unser Logo, den doppelköpfigen grünen Swap auf der Titelseite? Der die Schildchen trägt? Ja, *40. Jahrgang* Forth Magazin „Vierte Dimension“! Bei unserer derzeitigen durchschnittlichen Lebenserwartung schon ein halbes Menschenleben. Ich bin gespannt auf die vielen schönen forthingen Jubiläumsbeiträge, die ihr alle nun einsenden werdet. Für das erste Heft diesen Jahres sind schon schöne Sachen gekommen.

ANDREW READ schildert, wie er Forth für seine Himmelsbeobachtungen verwendet. Sein Credo: „Forth is no longer a protest movement!“ Und so zeigt er uns, wie er *Forth mit C verschmilzt*, um die Stärken beider Sprachen nutzen zu können. Im *VFX Forth* gelingt das leicht.

Danach geht es rechenintensiv weiter. STEPHAN BECHER hat eine *statische Typprüfung* in sein *StrongForth* eingebaut, damit sie dem Namen alle Ehre macht: Ein starkes Stück Forth.

Und auch das auf Steuerung spezialisierte *Fancy-Forth* von JÖRG VÖLKER ist nochmal komfortabler geworden. Nicht nur, dass es komplett auf Gleitkommaberechnungen aufgebaut ist, auch die einfache Verwendung von *lokalen Werten* macht es komfortabel. Und das bei klein gehaltenem Forthkern, der nach wie vor bequem in einfache MCUs passt. Die Kalkulationen sind hier komplett in Forth gehalten, klassisch, mit assemblierten Passagen.

Und Huckepack hat er auch die Meinung von ChatGPT zur Zukunft von Forth dazugegeben. Der Bot formuliert so schön ...

Dass FRANCOIS LAAGEL ein Board um die gute alte *HD63C09 CPU* herum gemacht hat, zunächst gefädelt, nun mit hübscher Platine, hatten wir ja berichtet. Nun hat RAFAEL DELIANO das Kit bekommen, aufgebaut und getestet. Lest selbst, was er dabei erfahren hat.

Schließlich beschreibt JOHANNES REILHOFER sein Leben mit Forth. Sein „Starting Forth“ ist schon fast ein Krimi.

Hier also wieder mein Appell: Schreibt mir bitte ein paar Zeilen darüber, wie ihr denn zum Forth gekommen seid.

So, und nun hoffe ich auf ein feines Forth-Sommertreffen.

Euer Michael



Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web-Seite des Vereins herunterladen.

<http://fossil.forth-ev.de/vd-2024-01>

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

Ulrich Hoffmann Kontakt: Direktorium@Forth-ev.de

Bernd Paysan

Gerald Wodni

100 Millionen Jahre Radio!

Knochenfische (Teleostei) entstanden im Jura vor etwa 150 bis 200 Millionen Jahren (Senckenberg Gesellschaft für Naturforschung).

Fische haben Elektrozyten in Reihenschaltung — *Elektroplax*. Diese elektrischen Organe liegen direkt unter der Haut. Es sind abgewandelte Muskelzellen, wie diese über reichlich Synapsen mit Nerven zum Gehirn verbunden und können so willentlich und rhythmisch codiert entladen werden. Was elektrische Felder um den Körper des Fisches bildet. Und sie haben in der Haut Rezeptororgane für diese Felder: Antennen. So orten sie aktiv das Gelände wie mit Radar, finden Strömungen im Wasser und „riechen“ sogar das Erdmagnetfeld. Und sie funken aktiv Signale zu ihren Artgenossen, spüren diese um sich herum — Radio! Ihre Antennen können noch Feldveränderungen von 0,1 mV wahrnehmen.

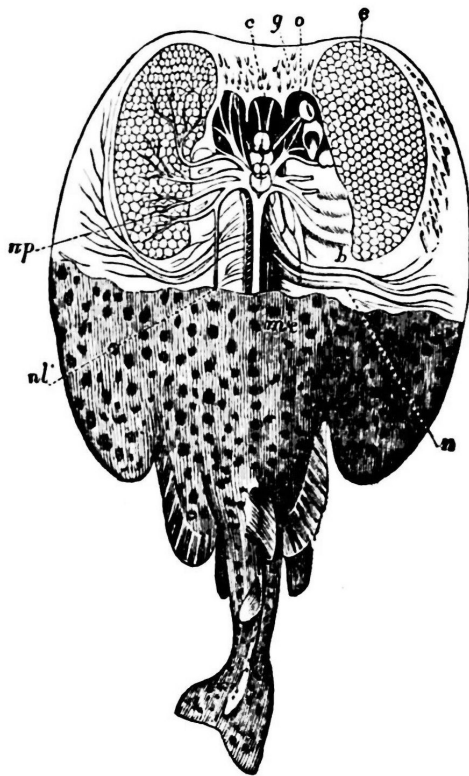


Abbildung 1: Elektroplax im Zitterrochen

Doch nicht nur die Knochenfische, auch Mikroorganismen, Blütenpflanzen, Insekten, Rundmäuler, Knorpelfische (Hai) und sogar einige Säugetiere haben Radio. Ist das nicht ein wunderschönes Beispiel dort bei Wikipedia, die Videoaufzeichnung des Elektrosignals eines männlichen Nilhechts bei der Werbung um ein Weibchen? Ich habe sehr gestaunt. mka

https://de.wikipedia.org/wiki/Elektrische_Orientierung

¹ VDE/VDI-Gesellschaft Mikroelektronik, Mikrosystem- und Feinwerktechnik. Heft 01-02-2024, S.10-11.

² China Software Industry Embedded Systems Associations.

Z79Forth

I am glad to announce that my kit project had reached production status ... The kit normally ships unassembled and I really think that putting it together is half of the fun.

I have some parts on order from Mouser that should reach my mailbox soon. If you are interested, there are a few options:

- assembled or “do it yourself”. The assembled flavour is tested in my lab. The DIY flavour is more fun but riskier.
- ANS94 Core or 79STD compliant. I recommend ANS94.

Should you be interested, I would require the following from you:

- your option preferences.
- a payment of 110 euros (via electronic fund transfer) to the following account:
- Holder:
Monsieur François Laagel 31, rue de l’Asile FR-68700 Cernay
IBAN: CH15 0029 2292 5951 0241 V
BIC: UBSWCHZH80A
- your postal address details.

Best regards.

Francois Laagel

f.laagel@ieee.org

<https://github.com/frenchie68/Z79Forth>

RISC-V Produktionszahlen

Beeindruckende Zahlen standen da neulich in der *Elektronik*, dem Organ der GMM¹, verfasst von Prof. Allan He, in seiner Eigenschaft als Vize-Direktor der SIA² und Gründer des Embedded-Software-Unternehmens *Beijing Microtech Research*, wie man in dem Heft zum Autor lesen konnte.

Im Jahre 2022 seinen weltweit „mehr als 10 Milliarden RISC-V-Chips“ produziert worden, die Hälfte davon in China. Und auf der RISC-V-Summit 2023 in Shanghai sei bekannt gegeben worden, dass wichtige Produktionsmeilensteine erreicht worden sind: 10 Millionen KI-Sprachchips von Chipintelli, 1 Million Low-Power-RISC-V-MCUs von XINSheng Technology (China Mobile) und weitere Superlative der Canaan-K210-Serie, dem Bouffalo Lab IoT-Chip und von HPMicro. Bis 2025 könne man es auf 80 Milliarden RISC-V-Chips bringen.

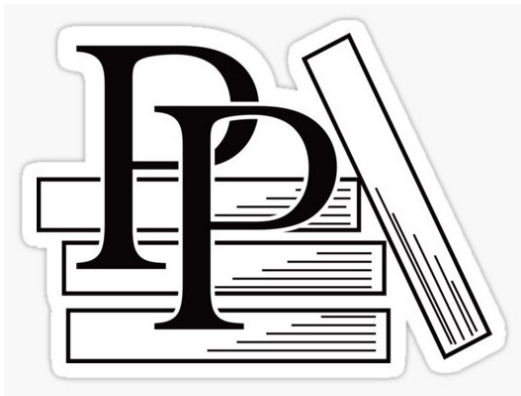
Bisher habe China die ausländischen Branchenführer studiert und ihnen nachgeeifert. Doch das habe sich geändert. Zitat Herr He:

„Durch die Fortschritte in Wissenschaft und Anwendung ist das Land nun in ein neues Zeitalter eingetreten: Es kann sich mit den weltweiten Wettbewerbern auf Augenhöhe messen.“

Die Branche sei vom Auftragslieferanten für Hardware und Anwendungssoftware zum Innovator von Prozessorkernen und Systemsoftware geworden, wobei drei Trends zu beobachten seien: Ki-Endgeräte, eine florierende Prozessorbranche um den RISC-V und eingebettete Software mit mehreren Betriebssystemen: Windows für die Oberflächen, Linux für die Netzwerke und Kommunikation sowie RTOS für Echtzeitleistung und Zuverlässigkeit.

Und es gäbe eine Zusammenarbeit von Wissenschaft und Industrie. Denn die eingebetteten Systeme seien der „Konvergenzpunkt mehrerer ... Disziplinen, darunter Informatik, Elektronik und Informationstechnik, Chipdesign, IoT und KI.“

mka



PragProg: Sieben obscure Sprachen

DMITRY ZINOVIEV, Professor für Informatik an der Suffolk University in Boston, interessiert sich leidenschaftlich für moderne C- und Python-Programmierung, komplexe Netzwerkanalyse, computergestützte Sozialwissenschaften, digitale Geisteswissenschaften, Computersimulation und -modellierung sowie Softwarearchäologie und Retrocomputing. Und er schreibt bei *PragProg* Bücher darüber: „Sieben obscure Sprachen — Wiederentdeckung der Werkzeuge, die die Zukunft gebaut haben“³.

“Immerse yourself in the intricate world of forgotten programming languages with *Seven Obscure Languages in Seven Weeks*. This comprehensive guide serves as a bridge to understanding and revitalizing legacy code, offering invaluable insights into the evolution of programming. With hands-on tutorials spanning languages from Forth and Simula to SNOBOL and m4, readers are equipped to maintain older systems and gain a broader perspective on problem-solving techniques. Whether you are a seasoned developer, a software historian, or just curious about the roots of modern coding, this book illuminates the rich tapestry of programming’s past and sheds light on its present and future.”

³ *Seven Obscure Languages in Seven Weeks — Rediscovering the Tools That Built the Future.*

Es ist neulich erschienen als eBook, Beta-Version, in der Serie *Seven in Seven* des *Pragmatic Bookshelf*. In dieser Serie werden dem Leser jeweils „in sieben Wochen“ Kenntnisse vermittelt über verschiedene Computersprachen oder Techniken wie “Modern Databases and the NoSQL Movement ... Mobile Apps ... Concurrency Models (leverage multiple cores, handle thousands of users and terabytes of data, and continue working in the face of both hardware and software failure.) ... Web Frameworks”.

“*PragProg* authors are everywhere. Some *PragProg* authors work on the cutting edge, pushing languages, architecture, and management forward at the most influential tech companies around the globe. Some *PragProg* authors are university professors, helping the next generation of computer scientists develop the foundations they’ll need to build the future. Some *PragProg* authors are creating the languages and libraries that are generating all the buzz and taking the development community by storm. And some *PragProg* authors are traveling the world, sharing their knowledge at conferences, meetups, and private training sessions ...”

Solche Leserhinweise sind immer willkommen. Lieber Carsten, danke für diesen Hinweis.

mka

<https://pragprog.com/titles/dzseven/seven-obscure-languages-in-seven-weeks/>

Forth wieder in Führung: Kleinste Programmiersprache auf dem PC

In der 4d2020-04 berichtete BERND PAYSAN vom *sectorForth*, einem von CESAR BLUM entwickelten Forth, welches in die 512 Byte des Bootsektors eines IBM-kompatiblen PC passt. Das war eine enorme Leistung und ein Beispiel, wie flexibel Forth als Sprache ist.

“Challenge Accepted” dachte sich wohl Programmierkünstlerin JUSTINE TUNNEY und schrieb ein Lisp, welches mit nur 436 Byte nochmals kleiner als *sectorForth* ist.

SectorLisp: <https://justine.lol/sectorlisp2/>

Schaut euch auch mal ihr *cosmopolitan libc-Projekt* an: <https://justine.lol/cosmopolitan/index.html>

Geht es noch kleiner? Ja, es geht. Diesmal wieder mit Forth: *milliForth*. Mit nur 380 Bytes ist dieses Forth von FUZZBALLCAT um 56 Bytes schlanker als *sectorLISP*.

<https://github.com/fuzzballcat/milliForth>

Mal schauen, ob es jemand schafft, diesen Wert noch zu unterbieten.

mka

Object Messaging

Neulich aus dem Netz gefischt:

“Don’t be fooled into thinking that objects are the most important things in object oriented programming languages. They are not. ALAN KAY, who coined the term *object orientation* has said that objects were not the most important idea and he regrets putting the word ‚object‘ in the name.

So what IS the most important thing in object orientation? It’s messaging!”

Aus: Code with HUW — Object Orientation is NOT about Objects. mka

<https://www.youtube.com/watch?v=lmAarc0Zhq4>

HH:MM:SS

Diese Perle fand ich neulich bei der Lektüre von MARC PETREMANNS neuem Buch [1]:

```
: :##
# 6 base !
# decimal
[char] : hold
;
: .hms ( n -- )
<# :## :## # # #> type
;
4225 .hms \ display: 01:10:25
```

Dazu sagte er dort:

“... I love this example because, to date, NO OTHER PROGRAMMING LANGUAGE is capable of achieving this HH:MM:SS conversion so elegantly and concisely.” ([1], S.29)

Gleich mal sehen, ob das auch in Gforth funktioniert — hm, ganz so nicht. Das liegt an der Eigenart vom Gforth, die Konversion der Zahl in eine Ziffer in einem *verschachtelbaren Puffer* zu machen. Der wird mit <<# und #>> definiert. Und die Zahl muss doppeltgenau angeliefert werden, was mit s>d bewerkstelligt werden kann. Also geben wir das noch dazu und schon klappt perfekt.

```
vocabulary test
test definitions
: :##
# 6 base !
# decimal
[char] : hold
;
: .hms ( n -- )
s>d
<<# :## :## # # #> type #>>
```

```
;
words cr cr
\ 4225 .hms \ display: 01:10:25
```

Der Rahmen `vocabulary ... words` ist dabei nur meiner Bequemlichkeit geschuldet. Denn so sehe ich die neu kompilierten Worte ohne den ganzen langen Rest an `forth words` des Gforth. mka

[1] The great book for ESP32forth (Version 1.13 — 15. November 2023)

<https://github.com/MPETREMANN11/ESP32forth>

Wiege des Forth

So bezeichnete KEVIN APPERT neulich das Observatorium am Kitt Peak⁴. CHARLES „CHUCK“ MOORE fasste dort seine damaligen Programmier-Tools zusammen zu dem, was dann zu Forth wurde.

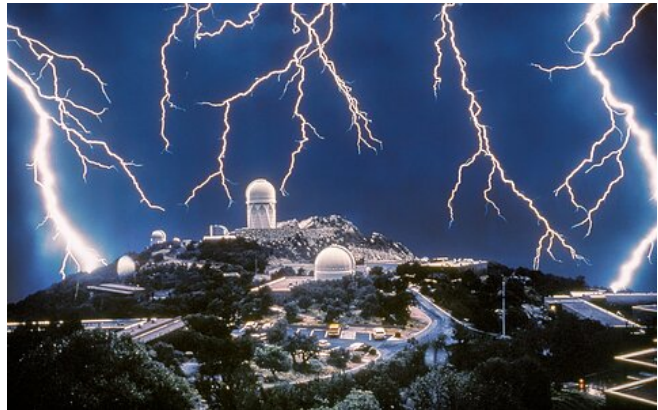


Abbildung 2: Gewitter am Kitt Peak

Nachgezeichnet wurde die Geschichte des Forth damals auf einer Präsentation beim Vintage Computer Festival West, 11. bis 12. Oktober 2003, im *Computer History Museum* in Mountain View, Kalifornien. Ein kaum funktionierendes Videoband davon ist wieder aufgetaucht, von dem BRUCE DAMER nun das Audio wiederhergestellt hat.

„... Der Audiopegel ist oft niedrig und es kommt zu erheblichen Geräuschen. Wir haben versucht, den Ton zu verbessern (aber er ist immer noch nicht großartig).“ [Dennoch hörensenswert! Danke, Carsten, für den Link.]

Forth — David Jaffe, C. H. Ting, Kevin Appert, Dwight Elvey; by Bruce Damer. Publication date 2003–10–11
Audio ins Internet Archive gestellt am 2023–12–11 21:56:28.

https://archive.org/details/vcf2003_forth

Durch das Archiv zu stöbern, ist übrigens sehr gefährlich! Es zieht einen tiefer und tiefer hinein ... mka

⁴Der Kitt Peak (Ioligam in der Sprache der Tohono O’odham) ist ein 2097-m-hoher Berg in Pima County im US-Bundesstaat Arizona. (Wikipedia)

Astroimaging in FORTH: Data Formats and Computations

Andrew Read

In the two previous articles we used Forth to control a telescope mount through a TCP/IP connection and a CMOS camera through the vendor C-language API. We took advantage of the Forth's special virtue: interactively "speaking" with each hardware device in its native command protocol (discovering its possibilities and developing an engineer's intuition for working with it) then building a domain application language integrating control of all hardware at a comfortable level of abstraction for typical use.

Figure 1 is an image of the Southern Magellanic Cloud taken during the commissioning phase in Chile by one of my technical collaborators. Figure 2 is a picture of an additional rig established locally as a "scale model" of the main equipment in Chile for developing operating techniques, equipment testing and problem solving.

Bugbears

To automate a night of astroimaging we need to be able to make astronomical calculations, such as the times of our target rising and setting, or the angular separation between the target and the moonlight. Such computations raise two familiar bugbears: formula expression in RPN¹ format and Forth floating point. As a matter of preference I would like to avoid both. Personally I find it impossible to review formula expressed in RPN format (although I know that isn't an issue for everybody reading this!). We could use a FORMula TRANslator of course, but then we are no longer programming in Forth but in another language beginning with "F". A second stack for floating point and endless interconversion with the (integer) stack? Maybe you don't agree, but to me that's too much of a mess. Is there another way? I argue "yes" and explain my approach.

Forth is no longer a protest movement! The C-language has all the capability we need for floating point computations and many well-tested libraries. So let's do the hard work in C.

Calculate Time and Position

The questions becomes, what datatypes shall we pass between C and Forth, how shall we interface our code, and what is the appropriate division of functionality between C and Forth? Astronomical calculations are mainly about time and position. How much time and position accuracy does astroimaging require? Scheduling the telescope barely requires accuracy of more than a minute, but to-the-second would be conventional. Astronomical positions are commonly expressed in degrees, arcminutes (one sixtieth of one degree), and arcseconds (one sixtieth of one arcminute). Typical high-resolution astroimaging uses cameras with several thousand pixels on each dimension and a typical image-scale of one arc-second per pixel. Arc-second precision, i.e. positioning the CMOS-sensor correct to one pixel in thousands, will be more than good enough for pointing the telescope.

We can adopt a three-integer datatype on the stack for times and celestial positions:

```
HH MM SS  
DEG MM SS
```

With (almost) boundless 32-bit integer cells, this datatype can be compressed to a single integer cell with no loss of information. It's just the total number of seconds:

$$SS + 60 * (MM + (60 * HH))$$

I've chosen to call this datatype a *finite fraction*, whether the single integer or three integer format. The key insight is that finite fractions generalize beyond times and positions. The theory of partial fractions tells us that any fractional quantity expressed as $\frac{x}{60*60}$ has a unique alternative representation in the format $x_1 + \frac{x_2}{60} + \frac{x_3}{3600}$.

Here are my operators for converting between the two formats:

```
~~~ ( x -- x1 x2 x3 )  
~ ( x1 x2 x3 -- x )
```

We can save and retrieve a calendar date in a single cell like this:

¹ Reverse Polish Notation


```
2024 02 06 ~ ( 7286526)
7286526 ~~~ ( 2024 2 6)
```

The single integer form in this case is not a meaningful quantity except as a compressed version of the three integer form, but it is a more convenient format for storage and handling.

Mixed-Language Approach

As an illustration let's convert the position of a target in local equatorial coordinates (hour angle and declination) to horizon coordinates (altitude and azimuth) — the computation also requires the latitude of the observer. Here is the VFX Forth code:

```
Extern: void "C" EQtoHZ_ext(int H, int dec, int lat, int * alt, int * az) ;
```

```
: EQtoHZ ( H dec -- alt az ) { | alt az -- }
  latitude ADDR alt ADDR az          \ use VFX locals for the pass-by-reference
  ( H dec lat &alt &az) EQtoHZ_ext ( --) alt az
;
```

A C-language function in a DLL (`EQtoHZ_ext`) takes the hour angle, declination and latitude, each to nearest-second precision in single integer finite fraction format. The computed altitude and azimuth are returned in two variables passed by reference with the function call. Calling a function like this isn't particularly forthlike, so we wrap it in `EQtoHZ` which simply takes the hour angle and declination on the stack and returns our altitude and azimuth on the stack — latitude is stored in a global value. VFX local variables are used to provide temporary storage for the pass-by-reference return values, avoiding the risks of global variables.

Dates for astronomical purposes are best expressed as the counted number of days (positive or negative) since or before a standard epoch. The chosen epoch typically doesn't matter as it is an equal offset in all calculations. We make the conversion between our calendar date and the count of days since the epoch in a C function. Now we have date and time datatypes:

```
: ~time ( hh mm ss -- T ) ~ ;
\ convert a time in hours, minutes and seconds to a single cell time finite fraction

: ~date ( yyyy mm dd -- D )
\ convert year, month, day to a single cell date
  ~ days_since_epoch \ C function
;

: ~date-time ( yyyy mm dd hh mm ss -- D T ) ;
\ D = number of integer days since the epoch
\ T = time of day to the nearest-second as a finite fraction
  ~time >R ~date R>
;
```

Computing Intervals

Computing intervals between times and dates or adding a duration to a time and date is the bread-and-butter of astroimaging computations. Date time arithmetic using finite fractions is clearly and transparently accomplished in Forth with integer arithmetic. We use it to add some power and flexibility to our interactive sessions at the Forth terminal:

```
24 0 0 ~ constant 24HOURS

: %clock ( T - D T )
\ adjust a time to within 0 - 24h plus an increment number of days
  24HOURS /mod swap
  dup 0< if
    24HOURS +
    swap 1- swap
  then
;
;
```

```
: +duration ( D1 T1 D2 T2 -- D3 T3)
\ add a duration express as a date-time to another date-time
\ D3 T3 = (D1 T1) + (D2 T2)
  rot swap          ( D1 D2 T1 T2)
  + %clock >R      ( D1 D2 +D R:T3)
  + + R>          ( D3 T3)
;
```

Testing

Let's not forget testing. I use the `simple-tester.f` package (EuroForth 2019). Examples:

```
Tstart
T{ 12 30 45 ~time }T 45045 ==
T{ 45045 ~~~time }T 12 30 45 ==
T{ 2017 10 12 ~date }T 2842 ==
T{ 2842 ~~~date }T 2017 10 12 ==
T{ 36 00 00 ~time %clock }T 1 12 00 00 ~time ==
T{ -26 00 00 ~time %clock }T -2 22 00 00 ~time ==
52 00 00 ~angle -> Latitude
T{ 05 51 44 ~time 23 13 10 ~angle EQtoHZ }T 19 20 04 ~angle 283 16 16 ~angle ==
T{ 19 20 04 ~angle 283 16 16 ~angle HZtoEQ }T 05 51 44 ~angle 23 13 10 ~angle ==
Tend
```

In conclusion, incorporating astronomical calculations into my astroimaging project has been an opportunity to develop a mixed-language approach for floating point calculations. To summarize:

1. A *finite fractions* data type is established to represent fractional quantities at fixed precision. Finite fractions have two alternative representations: three integers (explicit, for the user) and single integer (compact, for storage and handling). 60 is an obvious radix for time and date quantities but 32 or 100 might be better choices in other applications.
2. Simple arithmetic with finite fractions is done in Forth using integer computation on the stack.
3. Computations requiring floating point arithmetic are performed by C-language functions interfaced through a DLL. The datatype exchanged between Forth and C is a finite fraction in single integer format.

The algorithms for astronomical computations in this project are taken directly from PETER DUFFETT-SMITH and JONATHAN ZWART, “Practical Astronomy with your Calculator or Spreadsheet (English Edition)”. All C and Forth code is open-source and available at my GitHub account, “Anding”.

As always, I invite comment and collaboration at andrew81244@outlook.com.

Practical Astronomy with your Calculator or Spreadsheet (English Edition) Englische Ausgabe | von Peter Duffett-Smith und Jonathan Zwart | 6. Juli 2017

Link

github.com/Anding

<https://vfxforth.com/>

My previous articles on the topic in our magazine:

4d2023-01 — First Steps Towards an Astroimaging Control System in Forth

4d2023-02 — Astroimaging in Forth: Image File Format and Camera SDK

More topics from me:

4d2015-0304 — The N.I.G.E. Machine

4d2016-02 — Developing an Ethernet Controller for the N.I.G.E. Machine

4d2018-01 — Developing a minimal UDP/IP stack for the N.I.G.E. Machine



Figure 1: Southern Magellanic Cloud, imaged with the author's equipment in Chile



Figure 2: An additional rig functioning as a local “scale model” for developing and testing

Statische Typprüfung in Forth

Stephan Becher

Forth ist eine von wenigen höheren Programmiersprachen, die auf eine statische Typprüfung verzichten. Viele Programmierer sehen dies als ein besonderes Feature an, weil der Compiler nicht jeden Programmiertrick gnadenlos anmeckert. Andere wünschen sich eine statische Typprüfung, die zugleich einen Stacktiefen-Check einschließen würde. Die Typprüfung erlaubt dem Compiler, eine Anzahl typischer Programmierfehler zu erkennen, etwa fehlende oder überflüssige Parameter sowie unbalancierte Verzweigungen und Schleifen. Auch die falsche Anwendung von Worten, beispielsweise die Benutzung von + anstelle von D+ oder die Vertauschung von Parametern, könnte der Compiler noch vor dem ersten Testlauf erkennen.

Ein konsistentes Datentyp-System

Mit den Stackkommentaren verfügt Forth bereits über ein Mittel, die Stackeffekte von Worten zu beschreiben. Leider handelt es sich lediglich um Kommentare. Nicht jeder Programmierer macht sich die Mühe, Stackkommentare anzugeben. Und vor allem: Sie werden vom Compiler ignoriert.

Aus Stackkommentaren verpflichtende Stackdiagramme zu machen, wäre ein entscheidender Schritt in Richtung auf eine statische Typprüfung. Der erste Schritt besteht indes darin, ein konsistentes System von Datentypen zu entwerfen. Tatsächlich spezifiziert der Forth 2012 Standard eine kleine Anzahl rudimentärer Datentypen:

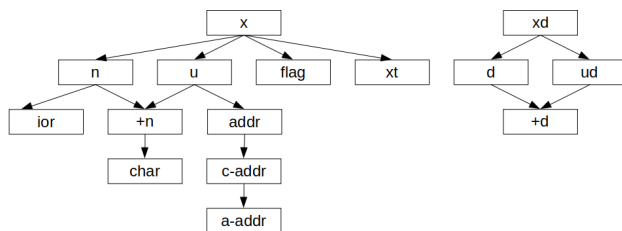


Abbildung 1: Datentypen gemäß Forth 2012

Datentypen, die auf `-sys` enden, sollen hier unberücksichtigt bleiben, ebenso wie der Datentyp `r` für Fließkommazahlen. An der Spitze der Hierarchie stehen die Datentypen `x` und `xd` als unspezifizierte Zellen bzw. Zellpaare. Davon sind untergeordnete Datentypen mit von oben nach unten zunehmender Spezifikation abgeleitet. Ein Character `char` ist etwa eine positive vorzeichenbehaftete Zahl `+n`, diese ist wiederum von vorzeichenbehafteten Zahlen `n` oder von vorzeichenlosen Zahlen `u` und diese von unspezifizierten Zellen `x` abgeleitet.

Diese Datentyp-Hierarchie ist der Ausgangspunkt. Unschön ist, dass `+n` und `+d` von gleich zwei übergeordneten Datentypen abgeleitet sind. Außerdem braucht ein verallgemeinertes System eine Vielzahl weiterer Datentypen. Insbesondere sollte es möglich sein, eigene Datentypen zu definieren und dafür entsprechende Worte bereitzustellen.

Deshalb soll hier ein streng hierarchisches Datentyp-System vorgeschlagen werden. Die Bezeichnungen der

Datentypen sind ungleich länger, dafür aber aussagekräftiger. Wegen der Länge der Namen wurde eine andere Darstellung gewählt:

```

single
  integer
    unsigned
    signed
  character
  address
  address
  logical
  flag
  token
  file
double
  integer-double
  unsigned-double
  signed-double
  
```

Auf Entsprechungen für die Datentypen `+n` und `+d` wurde verzichtet. Vorzeichenbehaftete und vorzeichenlose Datentypen wurden zusammen mit dem Datentyp `character` von dem Datentyp `integer` abgeleitet, der wiederum auf dem Datentyp für unspezifizierte Zellen, `single`, basiert. Dem gleichen Muster folgen die Datentypen für Zellpaare.

Aber warum sind Adressen keine vorzeichenlosen Zahlen? Der Grund ist, dass auf Adressen nur wenige arithmetische Operationen sinnvoll angewendet werden können. Der Compiler soll die Möglichkeit haben, illegale Adressoperationen wie Additionen zweier Adressen oder gar Multiplikationen zu erkennen. Zusätzlich wurde der Datentyp `logical` eingeführt, der für Bitfelder verwendet werden kann. Davon abgeleitet ist der Datentyp `flag`.

Stackdiagramme

Eines der elementarsten Wörter in Forth ist `drop`. Zusammen mit dem Stackdiagramm würde es wie folgt aussehen:

```
drop ( single -- )
```

Befindet sich auf dem Stack nun beispielsweise eine vorzeichenbehaftete Zahl vom Datentyp `signed`, könnte `drop` darauf angewendet werden, weil `signed` von `integer` und `integer` wiederum von `single` abgeleitet ist. Ein Wort gilt also immer auch für alle Datentypen, die direkt oder

indirekt von den Datentypen der jeweiligen Parameter abgeleitet sind. Eine separate Version von `drop` für `signed` wird nicht benötigt.

Allerdings kann `drop` nicht auf Zellpaare angewendet werden, denn die haben entweder den Datentyp `double` oder einen direkt oder indirekt davon abgeleiteten Datentyp. Um ein Zellpaar vom Stack zu entfernen, wird, wie in Forth 2012, ein anderes Wort benötigt:

```
2drop ( double -- )
```

Auch dieses Wort kann auf alle vom Datentyp `double` abgeleiteten Worte angewendet werden, also auf `integer-double`, `unsigned-double` und `signed-double`. Aber es geht noch einfacher. Es ist nicht unbedingt nötig, einen eigenen Namen, in diesem Fall `2drop`, zu verwenden. In einem Forth mit statischer Typprüfung können Worte überladen werden:

```
drop ( single -- )
drop ( double -- )
```

Befindet sich ein Zellpaar auf dem Stack, würde der Compiler feststellen, dass die erste Version von `drop` zu einer Typverletzung führen würde. Er durchsucht das Dictionary dann einfach weiter und stößt schließlich auf die zweite Version. Diese Version verursacht keine Typverletzung, und kann folglich kompiliert werden. Andere Worte lassen sich ebenfalls überladen, beispielsweise `+`, `D+`, `M+` und `F+`, aber auch `<`, `U<`, `D<`, `DU<` und `F<`. Allerdings entstünde daraus der Nachteil, dass ursprünglich für Forth 2012 geschriebener Quellcode geändert werden muss. Doch das wäre beim Umstieg auf ein System mit statischer Typprüfung ohnehin erforderlich, weil alle Worte mit einem Stackdiagramm versehen werden müssen.

Interessant ist die Tatsache, dass es mit diesen beiden Versionen von `drop` nicht möglich ist, aus einem Zellpaar eine einzelne Zelle mit dem niederwertigen Teil zu gewinnen. Um das zu erreichen, muss `d>s` verwendet werden:

```
d>s ( double -- single )
```

Indem `drop` überladen wird, wird das Wort `2drop` keinesfalls überflüssig. Denn die Version von `drop` für Zellpaare kann in einem System mit strenger Typprüfung nicht auf zwei einzelne Zellen angewendet werden. Dies wäre das Stackdiagramm von `2drop`:

```
2drop ( single single -- )
```

Sehen wir uns ein weiteres überladenes Wort an:

```
dup ( single -- single single )
dup ( double -- double double )
```

Auf jeden Datentyp würde eine dieser beiden Versionen passen, ausgenommen auf Fließkommazahlen, die vorläufig ausgeklammert werden sollen. Allerdings liegen die Dinge nicht ganz so einfach wie bei `drop`. Befindet sich auf dem Stack eine Zelle vom Datentyp `character`, würden nach der Kompilierung von `dup` zwei Zellen vom Datentyp `single` auf dem Stack landen statt zwei Zellen vom Datentyp `character`. Damit das funktioniert, müssen Datentyp-Referenzen eingeführt werden:

```
dup ( single - 1st 1st )
dup ( double -- 1st 1st )
```

`1st` anstelle eines Datentyps in der Liste der Ausgangsparameter bedeutet, dass der Datentyp nach der Kompilierung mit dem ersten tatsächlichen Datentyp in der Liste der Eingangsparameter identisch ist, im Beispiel also `character`. Erwartet ein Wort mehrere Parameter auf dem Stack, müssen die Schlüsselwörter `2nd`, `3rd` usw. benutzt werden:

```
swap ( single single -- 2nd 1st )
\ entspricht SWAP
```

```
swap ( single double -- 2nd 1st )
\ entspricht ROT
```

```
swap ( double single -- 2nd 1st )
\ entspricht ROT ROT
```

```
swap ( double double -- 2nd 1st )
\ entspricht 2SWAP
```

Von `swap` werden bereits vier überladene Versionen benötigt, um alle Kombinationen von Zellen und Zellpaaren abzudecken. Noch extremer wird es für `rot`, weil dafür insgesamt acht verschiedene Kombinationen von Datentypen zu berücksichtigen sind. Die vier Versionen von `swap` sind tatsächlich erforderlich, weil `rot` bzw. `rot rot` in den beiden mittleren Versionen von `swap` Datentyp-Verletzungen zur Folge hätten. Es befinden sich zwar drei Zellen, aber nur zwei Parameter auf dem Stack. Vier unterschiedliche Versionen müssen auch für `over`, `nip` und `tuck` bereitgestellt werden.

Betrachten wir nun das Forth 2012 Wort `+`:

```
+ ( integer integer -- 1st )
```

Ob vorzeichenbehaftet oder vorzeichenlos, spielt bei `+` keine Rolle. Wichtig ist, dass das Ergebnis den gleichen Datentyp hat wie der erste Parameter. Ein Zahlenwert, addiert auf einen Character, ergibt also wieder einen Character. Auf Datentypen `address`, `logical`, `token` und `file` kann diese Version von `+` indes nicht angewendet werden, weil sie nicht von `integer` abgeleitet sind. Bei Bitfeldern und Flags, Execution-Tokens und Files ist das nachvollziehbar, denn bei diesen Datentypen ergibt eine Addition keinen Sinn. Aber bei Adressen? Ja, es ist tatsächlich notwendig, eine überladene Version von `+` für Adressen zu definieren:

```
+ ( address integer -- 1st )
```

Die Semantik ist exakt die Gleiche wie die der Version für Zahlen. Was es hingegen nicht gibt, ist eine Version, mit der zwei Adressen addiert werden:

```
+ ( address address -- 1st )
```

Die Addition zweier Adressen wird also gewissermaßen vom Compiler unterbunden. Es ist lediglich möglich, eine Zahl zu einer Adresse zu addieren. Das Ergebnis ist eine neue Adresse.

Um sauber mit Adressen umzugehen, reichen die bisher gezeigten Möglichkeiten, Stackdiagramme zu definieren, nicht aus. Das wird anhand des Forth 2012 Wortes `@` deutlich:

```
@ ( address -- single )
```

Das Stackdiagramm ist insofern unbefriedigend, als das Ergebnis immer eine unspezifische Zelle ist. Wir würden stattdessen erwarten, dass `@`, angewandt etwa auf die Adresse einer Variable, eine Zelle liefert, deren Datentyp dem der Variable entspricht. Beispielsweise sollte

```
base @
```

nicht einen Wert vom Datentyp `single`, sondern einen vom Datentyp `unsigned` liefern. Die Lösung besteht darin, Adressen zu verwenden, in denen der Datentyp des Wertes, auf den sie zeigen, enthalten ist. `base` gibt also nicht einen Wert vom Datentyp `address` zurück, sondern vom zusammengesetzten Datentyp

```
address -> unsigned
```

Auch das Stackdiagramm von `@` muss angepasst werden:

```
@ ( address -> single -- 2nd )
```

Auf unspezifizierte Adressen vom Datentyp `address` kann mit `@` nicht zugegriffen werden. Die Adresse muss zwingend einen zusammengesetzten Datentyp haben. `2nd` ist eine Referenz auf den zweiten Datentyp des Stackdiagramms, also auf `single`. Wird `@` auf eine Adresse vom Datentyp `address -> unsigned` angewandt, ist das Resultat somit tatsächlich vom Datentyp `unsigned`. Selbstverständlich ist `@` überladen:

```
@ ( address -> single -- 2nd )  
  \ entspricht @
```

```
@ ( address -> double -- 2nd )  
  \ entspricht 2@
```

```
@ ( caddress -> single -- 2nd )  
  \ entspricht C@
```

```
@ ( caddress -> signed -- 2nd )  
  \ entspricht C@ mit sign extension
```

```
@ ( caddress -> flag -- 2nd )  
  \ entspricht C@ mit sign extension
```

Sofern diese Versionen im Dictionary so angeordnet sind, dass die unteren Versionen zuerst gefunden werden, wählt der Compiler die Version von `@` aus, die zum zusammengesetzten Datentyp der Adresse passt. Dank der beiden letzten Versionen kann er sogar vorzeichenbehaftete Zahlen und Flags aus dem Speicher holen, die die Größe eines Characters haben. Vergleichbare Worte sind in Forth 2012 nicht spezifiziert.

Mit zusammengesetzten Datentypen kann `@` typsicher kompiliert werden. Das Gleiche gilt für `!`, doch liegt der Fall hier wiederum ein wenig anders:

```
( single address -> 1st -- )
```

Der entscheidende Punkt ist, dass in diesem Stackdiagramm eine Datentyp-Referenz innerhalb der *Eingangsparameter* auftritt. Die Referenz hat hier eine etwas andere Bedeutung als in den Ausgangsparametern. `!` kann nur kompiliert werden, wenn die Referenz mit dem referenzierten Datentyp identisch ist.

```
10 base !
```

würde also funktionieren, weil `10` den Datentyp `unsigned` hat und `base` den Datentyp `address -> unsigned`.

```
true base !
```

weist der Compiler hingegen als Datentyp-Verletzung zurück, denn `true` hat den Datentyp `flag` und passt somit nicht zum Datentyp der zusammengesetzten Adresse. Natürlich ist auch `!` überladen:

```
! ( single address -> 1st -- ) \ entspricht !  
! ( double address -> 1st -- ) \ entspricht 2!  
! ( single caddress -> 1st -- ) \ entspricht C!
```

Typsicheres Forth

Um ein typsicheres Forth-System nach diesem Vorbild zu implementieren, müssen vier Ergänzungen vorgenommen werden:

1. Es muss sowohl zur Laufzeit als auch zur Kompilierzeit bekannt sein, welche Datentypen die Werte haben, die sich auf dem Stack befinden.
2. Bei der Definition neuer Worte ist es erforderlich, das Stackdiagramm im Dictionary abzulegen.
3. Compiler und Interpreter müssen beim Durchsuchen des Dictionarys die Datentypen der Werte auf dem Stack mit den Stackdiagrammen der Worte im Dictionary abgleichen. Nur so können Typverletzungen aufgedeckt und die passende überladene Version ausgewählt werden.
4. Bereits existierende Worte können überladen und gegebenenfalls ergänzt werden. Diese Änderung ist optional, aber sinnvoll.

Zu 1.:

Um den Werten auf dem Stack Datentypen zuzuordnen, genügt es, eine aus einfachen und zusammengesetzten Datentypen bestehende Liste anzulegen. Die maximale Länge der Liste hängt von der maximal erwarteten Stacktiefe ab, wobei zu berücksichtigen ist, dass zusammengesetzte Datentypen mehrere Einträge in der Liste erfordern. Zusammengesetzte Datentypen können sogar aus mehr als zwei einfachen Datentypen bestehen, beispielsweise die Adresse einer Variablen, die selbst eine Adresse enthält.

Eine einzige Datentyp-Liste genügt jedoch nicht. Der Interpreter und der Compiler müssen über jeweils eigene Datentyp-Listen verfügen. Während der Kompilation eines Wortes werden beide Listen parallel benutzt. Kompilierte Worte beziehen sich auf die Datentyp-Liste des Compilers, während Immediate-Worte im Kontext der Datentyp-Liste des Interpreters ausgeführt werden. Zu

Beginn der Kompilation wird die Datentyp-Liste des Compilers mit den Datentypen der Eingangsparameter initialisiert, wie sie aus dem Stackdiagramm hervorgehen. Am Ende der Kompilation wird die Datentyp-Liste des Compilers mit den Ausgangsparametern verglichen. Sind beide Listen nicht identisch, liegt eine Datentyp-Verletzung vor.

Um die beiden Datentyp-Listen aktuell zu halten, müssen Interpreter und Compiler bei der Ausführung bzw. der Kompilation jedes Wortes die Listen aktualisieren. Die Eingangsparameter des Wortes werden aus der Liste entfernt. Die Ausgangsparameter werden hinzugefügt, wobei Referenzen auf die Eingangsparameter beachtet werden müssen.

Zu 2.:

Das Stackdiagramm jedes Wortes wird zu dessen festem Bestandteil. Um es zu erzeugen, muss die Semantik des Forth 2012 Wortes (geändert werden. Für Kommentare steht dann nur noch das Wort \ zur Verfügung. (könnte das nachfolgende Stackdiagramm parsen. Eine elegantere Lösung ist jedoch, das Stackdiagramm sich selbst erzeugen zu lassen. Die Namen aller Datentypen, dazu `1st`, `2nd`, `3rd`, `->`, `--` und `)`, sind normale Forth-Wörter, die dazu dienen, ein Stackdiagramm zu konstruieren. (muss in diesem Fall lediglich einige Initialisierungen durchführen und in den Interpreter-Zustand zurückschalten.) beendet die Erzeugung des Stackdiagramms und schaltet wieder in den Compiler-Zustand.

Zu 3.:

`find` muss beim Durchsuchen des Dictionaries nicht nur die Namen der Worte berücksichtigen, sondern auch feststellen, ob das Stackdiagramm eines Wortes zum Inhalt der jeweiligen Datentyp-Liste passt. Ist das nicht der Fall, gibt es keinen Match und die Suche wird fortgesetzt, um eine passende überladene Version des Wortes zu finden. Befindet sich beispielsweise an der obersten Position des Stacks ein Wert vom Datentyp `unsigned`, und findet `find` als Erstes die Version `dup (double -- 1st 1st)`, muss `find` dieses Wort verwerfen und weitersuchen. Erst wenn `find` auf `dup (single -- 1st 1st)` stößt, gibt es einen Match. Welche der beiden Datentyp-Listen `find` verwendet, hängt davon ab, ob das Wort interpretiert oder kompiliert wird, also von `state` und vom Immediate-Flag des jeweiligen Wortes.

Zu 4.:

Ein Datentyp-System bietet die Möglichkeit, Worte zu überladen. In fast allen höheren Programmiersprachen ist diese Technik üblich. Um etwa Fließkommazahlen zu addieren, schreibt man `+` und nicht `F+`. Überladung funktioniert auch in einem Forth-System mit strenger Typprüfung, vorausgesetzt, sämtliche überladenen Worte lassen sich anhand ihrer Eingangsparameter unterscheiden.

Manch einer wird die Überladung von Worten, ebenso wie ein Datentyp-System, als nicht *forth-like* ansehen. Die Vorzüge sind dennoch unbestreitbar. Der Compiler wählt automatisch die zu den Operanden passende Version eines Wortes aus. Es kann nicht länger passieren,

dass beispielsweise bei der Umstellung eines Wortes von Zell-Operanden auf Zellpaar-Operanden an einer Stelle vergessen wird, `+` durch `D+` zu ersetzen. Oftmals sieht der Quellcode sogar identisch aus, wie in diesem Beispiel:

```
: mean ( signed 1st -- 1st )
  + 2/ ;
: mean ( signed-double 1st -- 1st )
  + 2/ ; \ entspricht D+ D2/
```

Hier ist nun ein einfaches Beispiel, wie sich die Datentyp-Liste des Interpreters von Wort zu Wort verändert. Weil (nicht mehr für Kommentare zur Verfügung steht, muss der Backslash als Kommentarzeichen erhalten.

```
base \ ( -- address -> unsigned )
@ \ ( address -> unsigned - unsigned )
. \ ( unsigned -- )
```

Wie werden überhaupt Variablen wie `base` erzeugt, die eine Adresse mit einem zusammengesetzten Datentyp liefern? Mit

```
variable foo
```

kann das nicht funktionieren, denn diese Phrase gibt keinen Anhaltspunkt zum Datentyp der neuen Variablen. Ein Forth-System mit strenger Typprüfung kann nicht darauf verzichten, einen solchen Hinweis zu geben. Die einfachste Lösung besteht darin, Variablen grundsätzlich zu initialisieren:

```
13 variable foo
```

definiert eine Variable mit dem Datentyp `unsigned` und initialisiert sie zugleich mit dem Wert `13`.

```
true variable bar
```

definiert eine zweite Variable mit dem Datentyp `flag` und initialisiert sie mit dem Wert `true`.

Und woher weiß der Interpreter, dass `13` eine vorzeichenlose Zahl ist? Ganz einfach: Literale vom Datentyp `signed` müssen immer ein Vorzeichen haben. `+13` ist also ein Literal mit dem Datentyp `signed` und dem Wert `13`.

Nach den bisherigen Ausführungen sollte klar sein, dass auch das Wort `.` überladen ist. Es gibt Versionen für die Datentypen `single`, `signed`, `character`, `flag` und viele mehr. Die Version für `single` gibt eine vorzeichenlose Zahl aus, gilt also auch für die Datentypen `integer`, `unsigned` und `address`, sofern für diese keine separate Version von `.` definiert wird.

Verzweigungen und Schleifen

Eine wichtige Frage ist, was mit den Datentyp-Listen geschieht, wenn der lineare Kontrollfluss des Compilers durch Verzweigungen und Schleifen unterbrochen wird.

```
: char ( -- character )
  parse-name if @ else drop bl then ;
```

Im Anschluss an die Kompilierung von `parse-name` sieht die Datentyp-Liste des Compilers wie folgt aus:

```
address -> character unsigned
```

Das Immediate-Wort `if` kompiliert einen bedingten Sprung, der den Wert vom Datentyp `unsigned` konsumiert. `unsigned` wird von der Datentyp-Liste des Compilers entfernt. Damit der Zustand der Datentyp-Liste am Anfang des zweiten Zweiges wiederhergestellt werden kann, muss `if` eine temporäre Kopie davon anlegen. Am Ende des ersten Zweigs, also nach der Kompilierung von `@`, hat die Datentyp-Liste den Inhalt

```
character
```

`else` erzeugt davon seinerseits eine Kopie und kompiliert einen unbedingten Sprung. Anschließend stellt `else` den Zustand derjenigen Datentyp-Liste wieder her, die von `if` gespeichert wurde. Diese Kopie kann nun gelöscht werden.

Der zweite Zweig beginnt also mit den gleichen Datentypen wie der erste Zweig. Nach `drop b1` befindet sich wieder ein Wert vom Typ `character` in der Datentyp-Liste des Compilers. `then` vergleicht den aktuellen Zustand mit der Kopie, die am Ende des ersten Zweigs von `else` gespeichert wurde. Gibt es eine Differenz, liegt eine Datentyp-Verletzung vor. Andernfalls, wie in unserem Beispiel, setzt der Compiler seine Arbeit fort. `;` vergleicht den aktuellen Inhalt der Datentyp-Liste mit den Ausgangsparametern des Stackdiagramms. Auch hier würde eine Differenz zu einer Datentyp-Verletzung führen.

Was im Falle einer Schleife mit der Datentyp-Liste des Compilers geschieht, soll anhand der Definitionen von `#s` und `spaces` erläutert werden.

```
: #s ( integer-double -- 1st )
  begin # dup 0= until ;
```

`begin` erstellt eine Kopie der Datentyp-Liste des Compilers, weil später an diese Stelle zurückgesprungen wird. `until` kompiliert den Rücksprung und verbraucht das von `0=` erzeugte Flag. Anschließend vergleicht `until` den aktuellen Inhalt der Datentyp-Liste mit der von `begin` angelegten Kopie. Stimmen beide nicht überein, ist die Schleife nicht ausbalanciert.

```
: spaces ( signed -- )
  begin dup 0>
  while space 1-
  repeat drop ;
```

Die Konstruktion `begin ... while ... repeat` kombiniert eine Schleife mit einer bedingten Verzweigung. `while` kompiliert einen bedingten Vorwärtssprung, der ein Flag konsumiert, und legt anschließend eine zweite Kopie der Datentyp-Liste des Compilers an. Es ist die Aufgabe von `repeat`, sowohl die beiden Sprünge aufzulösen als auch die Konsistenz der Datentyp-Liste sicherzustellen. Weil von `repeat` aus ein unbedingter Rücksprung zum Anfang der Schleife erfolgt, muss der aktuelle Inhalt der Datentyp-Liste des Compilers mit der von `begin` angelegten Kopie übereinstimmen. Da es sich um einen unbedingten Sprung handelt, kann die Datentyp-Liste anschließend verworfen werden. Ihr Inhalt wird ersetzt durch die von `while` angelegte Kopie. Damit wird die Definition fortgesetzt.

Type-Casts

Der Eingangsparameter von `spaces` im letzten Beispiel ist eine vorzeichenbehaftete Zahl. Somit würde `find` dieses Wort nicht im Dictionary finden, wenn eine vorzeichenlose Zahl an oberster Stelle des Stacks liegt:

```
5 spaces
```

führt zu einer Datentyp-Verletzung. Das ist natürlich unerwünscht. Das Problem ließe sich lösen, indem `spaces` überladen wird:

```
: spaces ( unsigned -- )
  begin dup
  while space 1-
  repeat drop ;
```

Das ist jedoch gar nicht nötig. Wenn der Datentyp des Eingangsparameters `integer` ist, genügt eine einzige Version von `spaces`, denn diese Version könnte sowohl auf vorzeichenbehaftete als auch auf vorzeichenlose Zahlen angewendet werden:

```
: spaces ( integer -- )
  cast signed
  begin dup 0>
  while space 1-
  repeat drop ;
```

Das Immediate-Wort `cast` bewirkt einen sogenannten *Type-Cast*. Es ersetzt den Datentyp an der Spitze der Datentyp-Liste durch den explizit angegebenen Datentyp. In diesem Fall wird `integer` durch `signed` ersetzt. Die Definition von `cast` ist nicht allzu komplex, würde aber hier den Rahmen sprengen, weil darin mehrere Worte benutzt werden, die Datentyp-Listen manipulieren. `cast` kompiliert sogar automatisch `s>d` oder `d>s`, wenn die beiden Datentypen auf dem Stack unterschiedliche Größen aufweisen.

Mehrdeutige Stackdiagramme

Einige der in Forth 2012 spezifizierten Worte haben keinen eindeutigen Stackeffekt. Das prominenteste ist zweifellos `?dup`. Der tatsächliche Stackeffekt hängt von einem Wert ab, der erst zur Laufzeit bekannt ist. Damit passt `?dup` nicht in ein Forth-System mit statischer Typprüfung. Denn eine statische Typprüfung zum Zeitpunkt der Kompilierung setzt voraus, dass die vom Compiler geführte Datentyp-Liste an jeder Stelle eindeutig ist.

Auf der anderen Seite sind Phrasen wie `?dup if`, `?dup while` und `?dup until` durchaus kompatibel mit einem System statischer Typprüfung. Tatsächlich wird `?dup` fast immer zusammen mit `if`, `while` oder `until` verwendet. Ein Forth-System mit statischer Typprüfung würde somit auf `?dup` verzichten und stattdessen Worte wie `?if`, `?while` und `?until` zur Verfügung stellen.

Ebenfalls verzichtet werden muss auf `pick` und `roll`. Ein großer Verlust wäre das indes nicht, denn die Verwendung dieser Worte ist fast immer mit schwer zu lesendem

Quellcode verbunden. Benutzt man Locals, werden `pick` und `roll` überflüssig.

Schwieriger ist die Situation bei `execute`, denn für dieses Wort gibt es keinen Ersatz. Der Stackeffekt von `execute` hängt davon ab, welches Execution-Token sich zur Laufzeit auf dem Stack befindet. Zur Kompilationszeit ist dieses Token nicht bekannt. Um das Problem zu lösen, müssen sogenannte qualifizierte Execution-Token als neue, von `token` abgeleitete Datentypen eingeführt werden. Ein qualifiziertes Execution-Token ist ein Token, dessen Stackeffekt bekannt ist. Beispielsweise ließe sich das qualifizierte Token

```
(unsigned--)
```

definieren, das bei Ausführung mit `execute` den Stackeffekt (`unsigned --`) hat. Passend zum Datentyp jedes qualifizierten Tokens muss dann nur noch eine überladene Version von `execute` bereitgestellt werden, in diesem Fall

```
execute ( unsigned (unsigned-- ) -- )
```

Diese Version von `execute` hat einen eindeutigen Stackeffekt. Mit Type-Casts lassen sich qualifizierte Token aus unqualifizierten Token vom Datentyp `token` erzeugen.

Zusammenfassung

Bei den in diesem Artikel vorgestellten Techniken für ein Forth-System mit statischer Typprüfung handelt es sich nicht um reine Gedankenspiele. In *StrongForth* sind sie genau so umgesetzt. Allerdings sollte klar sein, dass ein System mit statischer Typprüfung nicht mit Forth 2012 kompatibel ist. Nicht nur, dass Stackdiagramme in einem festen Format zwingend vorgeschrieben sind, dass (nicht mehr für Kommentare zur Verfügung steht und dass gewohnte Worte auf einmal überladen sind. In vielen Fällen wird der Compiler zudem eigentlich korrekten Code zurückweisen, etwa wenn Datentyp-Verletzungen nicht auszuschließen sind oder Programmiertricks implementiert wurden. Oft genügen Type-Casts oder kleinere Anpassungen des Codes, um die Bedenken des Compilers zu zerstreuen. Die Erstellung des Quellcodes erfordert mithin mehr Sorgfalt. Im Gegenzug dürfte sich die Codequalität verbessern. Der Zeitaufwand fürs Debugging sinkt, weil viele Fehler bereits vom Compiler aufgedeckt werden.

Link

StrongForth 3.0: <https://www.stephan-becher.de/strongforth3>



Abbildung 2: ;-) — Der Gedanke für das Logo kam mir so beim Layout — und weil hier noch Platz war ... Wie findet ihr es? (Der Setzer. mk)

Locals in Fancy–Forth

Jörg Völker

Dieser Beitrag bezieht sich auf meinen Artikel „Tags auf dem Return-Stack“ im Heft 4d2022-02, in dem ich das Projekt Fancy–Forth vorgestellt hatte. Auf der Forth-Tagung 6. Mai 2023 sprachen wir erneut darüber — die Folien zu dem Thema sind im Anhang, sie zeigen euch die Essentials des Projektes. In meinem kleinen Vortrag zu den weiteren Vorhaben der Fancy–Forth-Entwicklung kamen dann die Tags für den Returnstack zur Sprache und, davon abgeleitet, eine simple und effiziente Methode, lokale Values zu implementieren und die Prinzipien eines Umbilical Forth Systems dennoch beizubehalten.

Konzept

Fancy–Forth (FF) wurde konzipiert für Embedded-Systems und relativ kleine Mikrocontroller. Es gibt zwei zentrale Unterschiede zu einem Standard–Forth auf einem PC:

1. In einem Mikrocontroller ist eine strikte Unterscheidung und Trennung zwischen RAM und Flash unvermeidlich. Das hat weitreichende Auswirkungen auf die Art der Implementierung.
2. In FF speziell ist das Forth-System zweigeteilt: Das Dictionary selbst und die für den Zugriff und die Verwaltung nötige Funktionalität liegt im Host, der eigentliche Interpreter und Compiler aber im Target. Wird der Host vom Embedded-System getrennt, gibt es kein Dictionary mehr im Target.

Aus 2. folgt nun unmittelbar, dass lokale Variablen oder Values nicht so wie im Standard üblich temporär zur Laufzeit im Dictionary angelegt werden können, weil das ja dann fehlt. Es muss also ein anderes Konzept her, denn `locals` sind ja ein grundsätzlich wünschenswertes Feature, keine Frage.

Verwendung

Variablen werden in Forth über ihre Adresse angesprochen und die Zuordnung vom Variablennamen zur Adresse erfolgt bei der Deklaration zur Übersetzungszeit. Lokale Variablen hingegen liegen zur Laufzeit irgendwo auf dem Stack, ihre Adresse ist zur Übersetzungszeit natürlich noch nicht bekannt. Bei den Values ist es ebenso.

Bei FF gibt es bereits das Konzept der *Stackframes* auf dem Return-Stack, bei dem jeder Frame mit einem Tag, einer Identifikations-Nummer, versehen ist.¹ Dieses Konzept kann man sehr einfach auch für lokale Variablen nutzen, indem man jedem Local einen eindeutigen Tag zuordnet. Bei der Programmierung möchte man natürlich die Locals mit einem Namen ansprechen anstatt ihrer Tag-Nummer. Die Zuordnung vom Symbol, also dem Namen, zu einem Tag kann technisch wie bei der Variablen-Deklaration erfolgen. Geht man jetzt noch einen Schritt weiter, macht man es gleich objekt-orientiert. Die Phrase:

```
value IchBinGlobal
```

allokiert sofort Speicher, merkt sich die Adresse und erzeugt ein Objekt² mit Verweis auf die passende Methodenliste. Und der Name dieses Values findet sich im Dictionary auf dem Host. Hingegen legt

```
local IchBinLokal
```

erstmal nur eine eindeutige Identifikations-Nummer (Tag) fest, um dann ebenfalls ein Objekt mit Verweis auf eine passende Methodenliste ins Dictionary aufzunehmen.

Derart vorbereitet, kann jetzt in jedem Wort so ein lokaler Value verwendet werden:

```
: OhEinTest ( nimmDas -- )
  >local IchBinLokal \ Angelehnt an >r legt >local einen Frame auf dem Return-Stack an
                    \ für den vorher definierten Tag
                    \ und schiebt auch den Top of Stack dort hin.
  IchBinLokal      \ Da es ein Objekt im Value-Stil ist,
                    \ holen wir so den Wert auf den Datenstack zurück,
  2 * 1 +          \ und machen irgendwas damit.
  to IchBinLokal   \ So speichert in den lokalen Value zurück.
  IchBinLokal .    \ Und gibt den Wert mal an die Konsole aus.
  ;                \ Und machen uns keine weiteren Sorgen: exit räumt alles auf!
```

¹ Siehe Artikel „Tags auf dem Return-Stack“ in 4d2022-02.

² Values haben bei FF keinen Initialisierung-Wert bzw. werden automatisch mit 0 initialisiert

Scope³ und Format

IchBinLokal kann zwar beliebig oft in beliebig vielen Worten so verwendet werden, der Scope zur Laufzeit ist allerdings (in der aktuellen Version von FF) auf das jeweilige Wort begrenzt. Ein `local` muss in dieser Version immer initialisiert werden. Technisch gesehen ist `>local` eine Methode, genauso wie `to` eine ist. Spezielle lokale Formate für Bytes und Worte wären natürlich möglich, sind aber auch genauso leicht verzichtbar. Wie immer gibt es einen Kompromiss zwischen Eleganz und Aufwand. So sind die Locals einfach so breit wie Stack–Werte.

Ein weiteres Beispiel der Verwendung:

```
local ErsterParameterN1 \ ungewöhnlich: die Deklaration schon zur Übersetzungszeit.
local ZweiterParameterN2
local IchBinLokal
: aufpassen ( N1 N2 -- ) \ Zwei Werte auf dem Stack; n2 liegt zu oberst.
  >local ZweiterParameterN2 \ >local arbeitet den Stack ab, "last in, first out".
  >local ErsterParameterN1 \ Also obacht mit der Reihenfolge!
  47 >local IchBinLokal \ muss für seinen Zweck initialisiert werden, hier mit 47.
  irgendwas ;
```

Begrenzte Menge an Locals

Die Tags sind bei Fancy–Forth nur ein Byte groß und damit theoretisch gut für über 250 Locals. In der Praxis dürfte man selten über 4 Stück verwenden, vermute ich einmal. Da ist dann auch die Tatsache, dass die Locals zur Laufzeit anhand ihrer Tags auf dem Return–Stack linear gesucht werden müssen nicht so tragisch. Damit bei der Deklaration mehrerer Locals die Tags automatisch hochgezählt werden, ist eine globale Variable vonnöten. Unangenehmer Nebeneffekt für die Implementation ist hierbei: Ein solcher Marker muß neben `here` und `there` jetzt auch den Wert des aktuellen letzten Tags speichern.

Anhang

Es folgen die Folien meiner Präsentation „Local Values super simple in Fancy–Forth“ anlässlich der Forth–Tagung 2023 am 06.05., welche als Video–Konferenz abgehalten worden war. Ich hoffe, verständlich gemacht zu haben, wie Locals in Fancy–Forth, dem *Umbilical Forth System*, funktionieren.

Projekt FancyForth, rekapituliert:

Forth “entschärfen”:

- Speicher und Rechenleistung sind heutzutage reichlich vorhanden
- Beispiel STM8: preiswerter 8-Bit Prozessor, 10x schneller wie Z80 / 6502
- Optimierung auf Geschwindigkeit und Speicherbedarf ist oft unnötig
- Deshalb andere Entwicklungsziele:

Programmierkomfort, Betriebssicherheit, Fehlertoleranz

- 32-Bit FlexiFloat Floating-Point für eine Komfort-Arithmetik
- Tags auf dem Return-Stack für eine bessere Fehlertoleranz

³ Gültigkeit

Projekt FancyForth: Tags auf dem Return-Stack

Problem “dummes” EXIT bzw. “;”

- Erwartet gültige Rücksprungadresse
- Stürzt in allen anderen Fällen kommentarlos ab
- Fehler deshalb nicht selten schwer zu lokalisieren

“Intelligentes” EXIT

- Entfernt Daten von >R automatisch, R> DROP kann entfallen
- Entfernt Daten von DO automatisch, UNLOOP kann entfallen
- Erlaubt 2EXIT über zwei Worte idR. ohne Aufräumen des Return-Stacks

Projekt FancyForth: Tags auf dem Return-Stack

Bisher: Vier verschiedene Tags

1. COLON (:) für Rücksprungadressen
2. >R für allgemeine Daten auf dem Return-Stack
3. DO für Schleifenzähler und Limit
4. Für einen lokalen Value X (alter Ansatz)

Projekt FancyForth: Locals

Neu dazu: Tags für lokale Values

- Locals werden global “vor”deklariert
- Jedes Local hat systemweit einen eindeutigen 8-Bit Tag
- Ein vordefiniertes Local kann dann in beliebig vielen Worten verwendet werden

Projekt FancyForth: Locals, erster Ansatz

```

value ichBinGlobal \ allokiert sofort Speicher, merkt sich die Adresse
                  \ und erzeugt ein Objekt mit Verweis auf eine passende Methodenliste
local ichBinLokal  \ legt erstmal nur eine eindeutige Identifikations-Nummer (Tag) fest
                  \ und erzeugt ein Objekt mit Verweis auf eine passende Methodenliste

\ Derart vorbereitet, kann jetzt in jedem Wort ein lokaler Value verwendet werden:

: ohEinTest ( nimmDas -- )
  >local ichBinLokal \ angelehnt an >r wird so ein Frame auf dem Return-Stack angelegt
                  \ mit dem vorher zugeordnetem Tag
                  \ und gleich auch der Top of Stack dort abgelegt
  ichBinLokal      \ da es ein Objekt im Value-Stil ist, holen wir so den Wert
  2 * 1 +          \ ... machen irgendwas damit
  to ichBinLokal   \ speichern so in den lokalen Value zurück
  ichBinLokal .    \ geben vielleicht mal aus
  ;                \ und machen uns keine weiteren Sorgen: exit räumt alles auf!

```

Projekt FancyForth: Locals, etwas schöner

```
}; immediate      \ Dummy

: { ( --)
    0 begin
    ' dup [ ] <> while
    >r 1+ repeat \ exe-token auf R-Stack, mitzaehlen
    drop begin \ } ignorieren
    dup 0> while \ rueckwaerts
    1-
    [ ] >local compile, \ methode
    r> compile, \ objekt
    repeat drop ; immediate

\ kostet gerade einmal 40 Bytes fuer 24 Words -> ca. 1.7 Bytes/Word

local a
local b
local c \ Vorab-Definition bleibt erforderlich

: test { a b c }
a . b . c . ;

\ 1 2 3 test 1 2 3 OK 0 :-)
\ : test { a b -- u } wäre dann allerdings wesentlich aufwändiger
```

Projekt FancyForth

Programmiersprache Forth. Zukunftsaussichten?

ChatGPT

Forth ist eine Programmiersprache, die bereits seit den 1970er Jahren besteht und insbesondere in der Embedded-System-Entwicklung und in der Steuerungstechnik eingesetzt wird. Es gibt nach wie vor eine aktive Gemeinschaft von Forth-Entwicklern und -Anwendern, die die Sprache weiterentwickeln und pflegen.

Allerdings hat Forth in den letzten Jahren an Bedeutung verloren, da andere Programmiersprachen wie Python, JavaScript, Java und C++ immer populärer wurden. Dies liegt unter anderem daran, dass diese Sprachen besser an moderne Entwicklungsumgebungen, Bibliotheken und Frameworks angepasst sind und eine größere Gemeinschaft von Entwicklern und Anwendern haben.

Das bedeutet jedoch nicht, dass Forth keine Zukunft hat. Es gibt immer noch spezialisierte Anwendungen, in denen Forth aufgrund seiner Effizienz und Flexibilität Vorteile gegenüber anderen Sprachen hat. Außerdem gibt es immer noch eine aktive Gemeinschaft von Forth-Entwicklern, die die Sprache weiter verbessern und pflegen.

Insgesamt würde ich sagen, dass Forth in absehbarer Zeit keine große Verbreitung mehr finden wird, aber in spezialisierten Anwendungen weiterhin eine wichtige Rolle spielen wird.

Links

M. Anton Ertl, Automatic Scoping of Local Variables, EuroForth '94.

<https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Locals.html#Locals>

Wir berichteten des Öfteren über diese Entwicklungen in unserem Forth-Magazin „Vierte Dimension“:

J. Völker, Projekt Fancy-Forth, 4d2022-01.

M. Kalus, Präfixe, Spezifizierer — Partikel in Forth, 4d2021-03.

J. Plewe, Die unendliche Geschichte der lokalen Variablen, 4d1991-01.

H. Hansen, Simple local Variable, 4d1988-01.

Z79 Forth Reference Board V0.3

Rafael Deliano

Die Leiterplatte kommt unbestückt, mit fast allen benötigten Bauteilen.

Es gibt zwar eine beiliegende englische Anleitung für Montage und Inbetriebnahme, aber ohne Stromlaufplan. Die Stückliste ist in der Anleitung enthalten, aber etwas überdetailliert — lange Texte werden nicht gelesen. Der Stromlauf findet sich jedoch in der Projektbeschreibung auf *GitHub* als PDF. Dort ist auch ein Foto der bestückten Platine, das sehr hilfreich ist. Einen Bestückungsplan gabs noch nicht, die Platine hat aber einen Bestückungsaufdruck.

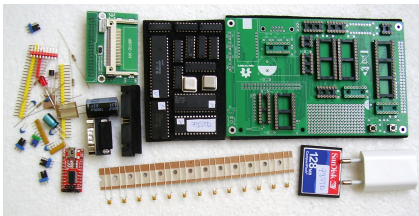


Abbildung 1: Anlieferungszustand

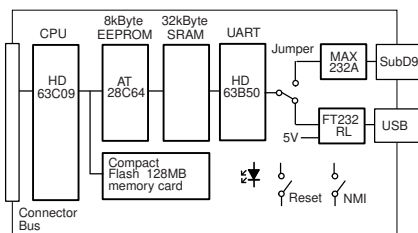


Abbildung 2: Blockschaltbild

Löten

Der LötKolben sollte zwar spitz sein, ist aber ansonsten unkritisch. Es gibt keine Masseflächen, die Wärme abziehen, so dass eine normale Wattzahl ausreichend ist. 1-mm-Lötzinn, klassisch verbleit bevorzugt. Da die Sockel schon unter Folie vorplaziert sind (Abb. 1), lötet man die als Erstes ein. Ein DIL16 für U2 fehlte. Dito ein Widerstand 47k. Der fand sich dann doch noch, aber in

Miniaturl-Bauform 0204, leicht zu übersehen. Für die 0207 Widerstände wäre die Angabe des Farbcodes, d. h. 10k = braun/schwarz/orange/gold, für manchen Anfänger nützlich. Bei den Kerkos¹ ist angenehm, dass es per Definition farbcodiert nur zwei Typen gibt: 100 nF (gelb) und 1 µF (blau). Nachteil war, dass die blauen 2,54 mm Raster hatten und auf 5,08 mm vorsichtig mit einer Zange aufgebogen werden mussten. Beim Stecker J1 ist die Orientierung auf dem Aufdruck nicht klar, aber ein Blick aufs Foto hilft. Da ist auch ein Hinweis in der Anleitung zu finden. Bei den Jumpers ist die Default-Position angenehmerweise im Bestückungsdruck markiert. Wenn man bevorzugt, die FTDI-Platine zu sockeln, sind dafür zwei zusätzliche 9-polige Buchsen nötig. Ferner entweder 4 Gummifüßchen an den Ecken, oder besser M3-Schrauben in 10 mm mit Muttern.



Abbildung 3: Benötigtes USB-Kabel

Kabel

Im Normalbetrieb wird das Board von einem Terminalprogramm auf dem PC über USB mit 115 kBAud 8N1 angesteuert (Abb. 4). Ein passendes USB-A- auf USB-mini-B-Kabel (Abb. 3) liegt jedoch nicht bei. Damit erfolgt auch die Versorgung mit 5 V, das ist die Default-Einstellung der Jumper. Die RS232-Option benötigt

ein Nullmodemkabel und das USB-Kabel wird dann mit dem beiliegenden Ladegerät verwendet.

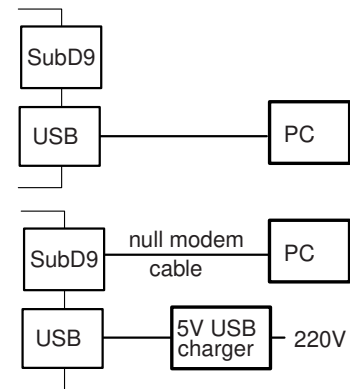


Abbildung 4: Anschlussvarianten — oben: Terminal und 5V-Versorgung per USB vom PC; unten: Serielle Schnittstelle und separate Stromversorgung.

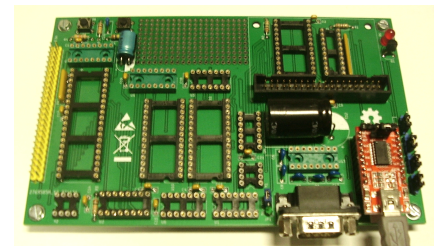


Abbildung 5: Test ohne ICs

Smoketest²

Erstmal Leiterplatte ohne ICs prüfen (Abb. 5). Auf dem RT232-Adapter den Jumper auf 5V stecken. Über USB mit Terminalprogramm verbinden. Am DIL-Sockel die 5V messen. Eine LED auf dem RT232 leuchtet dauernd, eine andere blinkt bei Tastendruck am PC. Bei älteren PCs muss man eventuell noch FTDI-Treiber installieren. Der verwendete Chip ist nicht Original-FTDI, aber ausreichend kompatibel.

¹ Keramikkondensatoren, kurz Kerkos genannt, bestehen aus dünnen Oxidkeramiksichten. Deshalb werden sie auch Keramik-Vielschicht-Kondensatoren genannt.

² Beim „Klempnern“ wird Rauch durch neu gelegte Rohre geführt, um evtl. Lecks zu finden, bevor Wasser in die Rohre gelassen wird. Bei der Reparatur von Holzblasinstrumenten wurde früher ein Ende des Instruments verstopft und auch hier Rauch hineingeblasen, um Löcher oder Risse im Material zu finden.

Bestücken und Test der Clock

Nächster Schritt ist das Bestücken der 74HCxx-ICs. Hilfreich bei Kelchkontakten ist ein „IC Pin Straightener“ (Abb. 6), z. B. Reichelt RW8920.



Abbildung 6: Pinbieger für ICs

Geprüft wird der phasenversetzte 4-MHz-Takt an der CPU, Pin 34 und Pin 35 (Abb. 7) sowie die 0,614 MHz an Pin 4 des UARTs (Abb. 8), wenn zeitweise Jumper 1 von 115k auf 38k gesteckt wird. Um bequem mit dem Oszilloskop zu kontaktieren, eignen sich die Drahtreste der Widerstände (Abb. 9). Für den GND-Clip empfiehlt sich eine Lötflanke an einer Ecken-Schraube (Abb. 10).

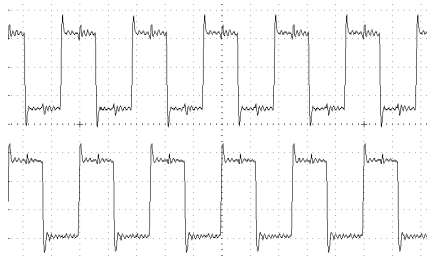


Abbildung 7: Takt CPU

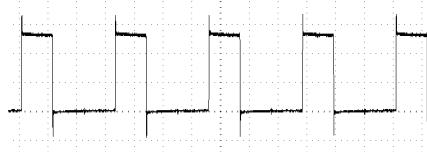


Abbildung 8: Baudrate UART

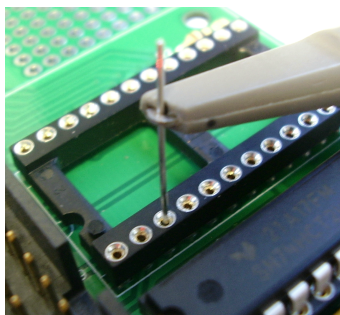


Abbildung 9: Kontaktierung Clock

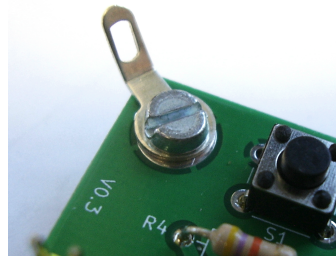


Abbildung 10: GND für Scope

LSIs

Die großen ICs sind schnell montiert. Auch, wenn man den Compact-Flashspeicher noch nicht bestückt hat, arbeitet das Forth im EEPROM bereits (Abb. 11, 12). Auf dem Non-Maskable-Interrupt liegt eine Debugfunktion (Abb. 13). Wenn man den CF-Speicher gesteckt hat, wird er auch erkannt (Abb. 14).

Montage und Inbetriebnahme war also völlig problemlos. Die Dokumentation des Musters war vorläufig und wird vermutlich noch überarbeitet.

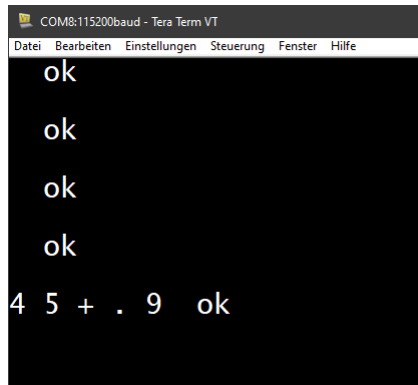


Abbildung 11: Erste Lebenszeichen

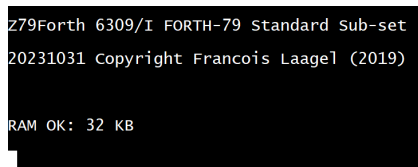


Abbildung 12: Resettaste ohne CF

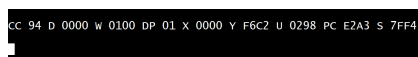


Abbildung 13: NMI-Taste

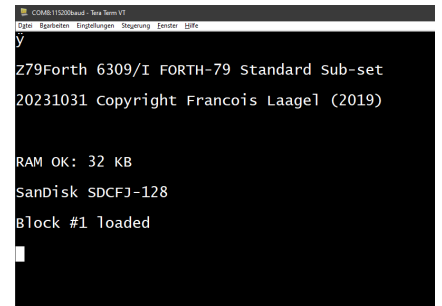


Abbildung 14: Resettaste mit CF

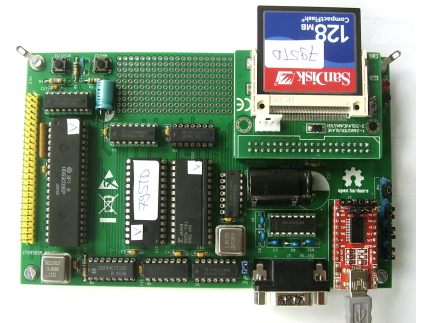


Abbildung 15: Bestückte Platine

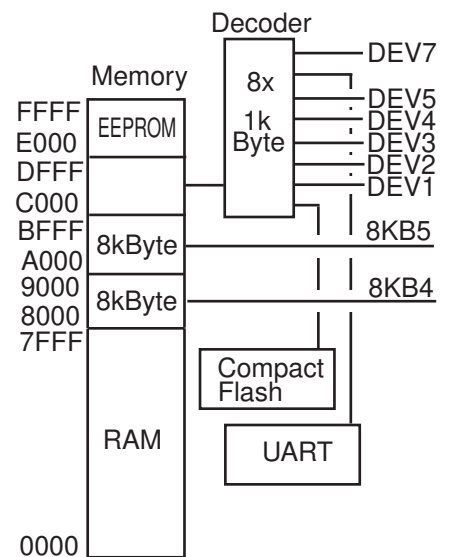


Abbildung 16: Memory-Map

Links

Ich vermute, die Dokumentation wird sich noch ändern.
<https://github.com/frenchie68/Z79Forth/blob/REL-ANS94/DSCF1020.JPG>

Die Entwicklungsgeschichte hat FRANCOIS LAAGEL hier festgehalten:
<https://github.com/frenchie68/Z79Forth/blob/REL-ANS94/README.md>

Mein „Starting Forth“

Johannes Reilhofer

Forth hat weitgehend mein Leben bestimmt.

Vor etwa 45 Jahren lernte ich DR. CHRISTIAN KUZNIA kennen. Er war der Chef einer 80-Mann-Gruppe bei SIEMENS, die weltweit untersuchte, was es denn technisch Neues gibt. Ein Fundstück war dabei Forth. Und Chris war es dann, der mir mit einem TRS-80 die ersten Grundlagen von Forth beibrachte.

Ich hatte vorher eine Firma¹ mit einem Stammkapital von 6000 Mark auf einen Umsatz von 70 Mio. gebracht — aber ... Der Vorstand der Röchling Industrieverwaltung hat mir dann für mein „Firmenimperium“ die Kaufsumme, 1 Markstück, über den Tisch geschoben. Und das war meine Rettung, ich war wieder schuldenfrei und konnte als 50-Jähriger neu beginnen. Und jetzt kommt Forth ins Spiel, und das gewaltig.

Schleifbrand

Das erste Beispiel für die Einmaligkeit von Forth ist unsere Schleifbrandverhinderung. Was ist das? Nun, Zahnräder, wie sie auch in LKW-Getriebe eingesetzt sind, müssen erst einmal Zähne bekommen. Dazu werden sie gefräst. Anschließend werden sie gehärtet und — jetzt kommt der kritische Prozess — sie werden geschliffen. Mit diesem letzten Arbeitsgang bekommen sie die ideale Form, Toleranz < 4 µm.

Das Schleifen ist leider der kritischste Arbeitsschritt. Hier kann es geschehen, dass Schleifbrand entsteht. Wenn es beim Schleifen zu einer Erwärmung von mehr als 450° Celsius irgendwo auf der geschliffenen Oberfläche kommt, hat man Schrott. Diese Zahnräder „sterben“ garantiert in der Garantiezeit. Als wir bei der ZF² begannen, existierte nur das Nitalätzen³. Die Zahnräder werden dazu kurz in Salpetersäure, dann in Schwefelsäure und zum Schluss in Natrium-Hydroxid getaucht.⁴ Ein furchtbarer Aufwand und eine riesige Sauerei.

Man wollte ein elektronisches Verfahren, um damit den Schleifbrand zu erkennen. Man muss dazu wissen, die Schleifbrandräder müssen im Schrott landen, würden sie im LKW landen — nicht auszudenken.

Wir, die REILHOFER KG, waren damals zwei Ingenieure und drei Studenten und wir hatten Forth. Wir machten der ZF den Vorschlag, das Ziel zu variieren: nicht den Schleifbrand erkennen, sondern ihn verhindern. Ohne Forth hätten wir es nie geschafft.

¹ Johne+Reilhofer

² Fahrradfabrik Friedrichshafen

³ Der Name „Nital“ ist eine Abkürzung von engl. „Nitric acid and alcohol“. Die übliche Ätzlösung für die Nitalätzung besteht aus Salpetersäure (HNO₃), Wasser (H₂O) und Ethanol (C₂H₅OH).

⁴ Chemische Prüfmethode zur Erkennung von Schleifbrand an metallischen Bauteilen. Das genannte Bad löst die Oberfläche des Werkstücks an und lässt die durch Schleifbrand verbrannten Bereiche in veränderten Graufarbtönen erscheinen — weiß: unbeschädigt; grau: geringe Schleifbrandbeschädigung; schwarz: starke Schleifbrandbeschädigung.

⁵ Kubisches Bornitrit

Warum Forth?

Vier Hochschulinstitute und etliche Firmen versuchten damals das Gleiche: den Schleifbrand erkennen. Niemand wusste, ob es charakteristische Emissionen dabei gibt. Wir wussten zwar, was wir verhindern wollten, aber nicht, wie. Und so fängt man an: Man schraubt in „Tatortnähe“ einen Beschleunigungsaufnehmer fest und hofft, dass irgendwelche Signalveränderungen mit dem Schaden korrelieren.

Immer, wenn noch etwas Neues erfunden werden muss, wenn man noch keine Ahnung hat, ob es überhaupt funktioniert, dann braucht man Forth. An der Schleifmaschine oder am Motorenprüfstand muss noch erfunden werden, die Lösung der Aufgabe ist noch unbekannt. Der Ingenieur tastet sich vor und dazu braucht er ein Werkzeug, das er kontinuierlich seinem aktuellen Erkenntnisstand anpassen kann. Diese Anforderung erfüllt nur Forth. Alle anderen Rechnersprachen setzen voraus, dass man schon weiß, was man will, dass das Problem schon gelöst ist, dass nur noch eine Werkzeugoptimierung stattfinden muss.

Wir haben uns angewöhnt, den Erfindungsprozess mit Forth zu betreiben. Wenn wir dann wissen „wie es geht“, folgt die Prozessoptimierung mit einer Hochsprache, die auch noch den Grafikprozessor einspannt. Wenn die Erfindung geschafft ist, geht es nur noch um Tempo und Kanalzahl.

Schleifbrand, was ist das?

Die Schleifscheiben haben das Profil der Zahnücke. Sie sind mit CBN⁵ beschichtet. CBN hat im Verhältnis zu Diamanten den Vorteil einer günstigeren Kristallform. Es sind Rhomben, die sehr schnell die Oberflächen abtragen. Bei LKW-Rädern werden typisch 150 µm von der Oberfläche weggeschliffen. Das kann natürlich streuen. Durch Härtingsverzüge können es schon mal 300 µm werden. Eine kritische Größe ist dabei das Spanvolumen, das ist der verbleibende Platz zwischen den CBN-Kristallen. Es kann geschehen, dass dieser Raum für das abgetragene Material nicht ausreicht, die Schleifscheibe fängt an zu drücken, statt zu schneiden. Die dabei entstehende

Reibungshitze beschädigt die Oberfläche des Zahnrades. Obwohl das meist nur einige Quadratmillimeter sind, reichen die aus, um das Zahnrad zu zerstören.

Die Schleifscheiben drehen sich typisch mit 6000 U/min, also mit 100 Hz. Die Maschinen heulen so laut, dass man gerne einen Geräuschschutz trägt. Jedes mal, wenn die Schleifscheibe in die nächste Zahnücke fährt, geht die Drehzahl lastbedingt zurück, um etwa 5–10%. Damit war für uns klar, die Drehzahl ist erstmal kein Indikator für die Schleifbrandentstehung. Aber was haben wir sonst noch? Wir haben breitbandig bis etwa 50 kHz die Beschleunigungen erfasst. Dabei fiel uns ein „Zirpen“ auf, das mit den Schäden korrelierte. Wir ermittelten die „ganz aktuelle Drehzahl“, indem wir die Zeit maßen, die pro Viertelumdrehung der Schleifscheibe verging (induktiv, 4 Inbusschrauben, 2,5 ms pro Viertelumdrehung, Zählfrequenz 1 MHz). Und dann entdeckten wir, dass die Drehzahl innerhalb einer Viertelumdrehung z. B. um 1 Promille einbrechen konnte. Wir rechneten die Energie, die frei wird, wenn eine rotierende Masse von über 50 kg in 2,5 ms um 1 Promille abgebremst wird. Wir kamen auf etwa 35 kW. Nachdem sich der Schleifbrand in den oberen 25 µm abspielt, war alles klar.

Und dann?

Das war es also — erst mal. Wir hatten jetzt den Indikator, wir erkannten die Entstehung von Schleifbrand,

aber noch nicht die Verhinderung! Dazu können wir nicht warten, bis der Indikator auftaucht. Wir müssen kurz davor in den Prozess eingreifen.

Unser „Forth-Erfindungsprogramm“ änderte sich jeden Tag, direkt an der Schleifmaschine. Wir maßen jetzt auch noch das Drehmoment, mit dem die Zahnräder gegen die Schleifscheibe drückten. Wir entwickelten Statistiken über die Härtungsverzüge, über die Nachbarverhältnisse zwischen den Zahnücken usw. Ich mache es kurz: Die ZF betreibt heute schleifbrandfrei, bei 30% Durchsatzerhöhung, alle Zahnradschleifmaschinen. In der Halle 10 in Friedrichshafen sparte man in einem Jahr etwa 12 Mio. €, weil man mehr und vor allem ohne Ausschuss produzieren konnte.

Auch der deutsche Schleifmaschinenhersteller profitierte von unseren Erkenntnissen; wesentliche Veränderungen waren die Folge. Und noch einmal: Patente brauchen Erfindungen. Wenn man nicht riskieren will, gegen die Zeit zu verlieren, braucht es Werkzeuge, die beim Erfinden helfen.

Wir sind heute beim Motoren- und Getriebeversuch und der Qualitätsendkontrolle von E-Antrieben und Verbrennungsantrieben weltweit vorne. Ohne Forth hätten wir es nicht geschafft. Langsam entwickeln kam nicht in Frage, wir hätten es finanziell nicht überlebt.

J.R., im Juli 2023



Abbildung 1: Schleifbrand

Forth-Gruppen regional

Bitte erkundigt euch vorab bei den Veranstaltern, ob die Treffen stattfinden.

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
bernd@net2o.de
Treffen: Jeden 4. Donnerstag im Monat um 19:00 auf <http://public.senfcall.de/forth-muenchen>, Passwort over+swap.

Hamburg **Ulrich Hoffmann**
Tel.: (04103) – 80 48 41
uho@forth-ev.de
Treffen alle 1–2 Monate in loser Folge
Termine unter: <http://forth-ev.de>

Ruhrgebiet **Carsten Strotmann**
ruhrpott-forth@strotmann.de
Derzeit keine Treffen.

Dienste der Forth-Gesellschaft

Nextcloud <https://cloud.forth-ev.de>

GitHub <https://github.com/forth-ev>

Twitch <https://www.twitch.tv/4ther>

µP-Controller-Verleih **Carsten Strotmann**
microcontrollerverleih@forth-ev.de
mcv@forth-ev.de

Spezielle Fachgebiete

Forth-Hardware in VHDL **Klaus Schleisiek**
microcore (uCore) Tel.: (0 58 46) – 98 04 00 8_p
kschleisiek@freenet.de

KI, Object Oriented Forth, Sicherheitskritische Systeme **Ulrich Hoffmann**
Tel.: (0 41 03) – 80 48 41
uho@forth-ev.de

Forth-Vertrieb **Ingenieurbüro**
volksFORTH **Klaus Kohl-Schöpe**
ultraFORTH Tel.: (0 82 66) – 36 09 862_p
RTX / FG / Super8
KK-FORTH

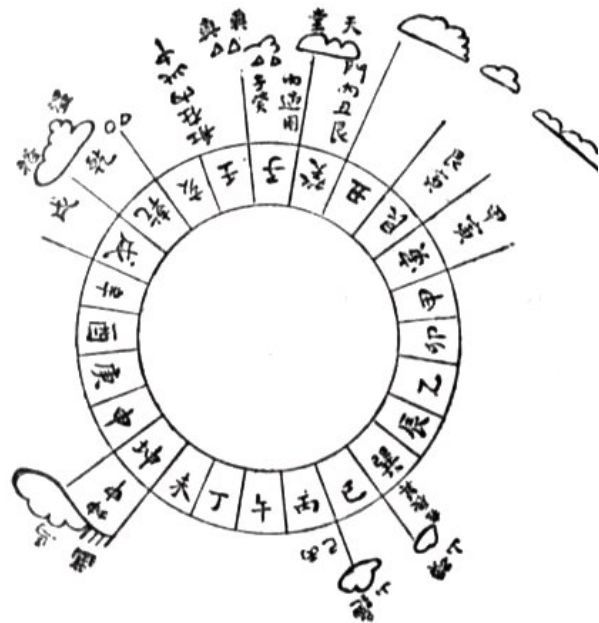
Termine

Donnerstags ab 20:00 Uhr
Forth-Chat net2o forth@bernd mit dem Key
keysearch kQusJ, voller Key:
kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*

Jeder 1. Montag im Monat ab 20:30 Uhr
Forth-Abend
Videotreffen (nicht nur) für Forthanfänger
Info und Teilnahmelink: E-Mail an wost@ewost.de

Jeder 2. Samstag im Monat
ZOOM-Treffen der Forth2020 Facebook-Gruppe
Infos zur Teilnahme: www.forth2020.org

Details zu den Terminen unter <http://forth-ev.de>



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

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

*Ob, und wenn, wo, die Tagung 2024 stattfinden wird,
ist noch völlig offen.*

Wünscht euch was!



„Was ist wichtiger“, fragte der große Panda,
„die Reise oder das Ziel?“

„Die Gesellschaft!“, sagte der kleine Drache.