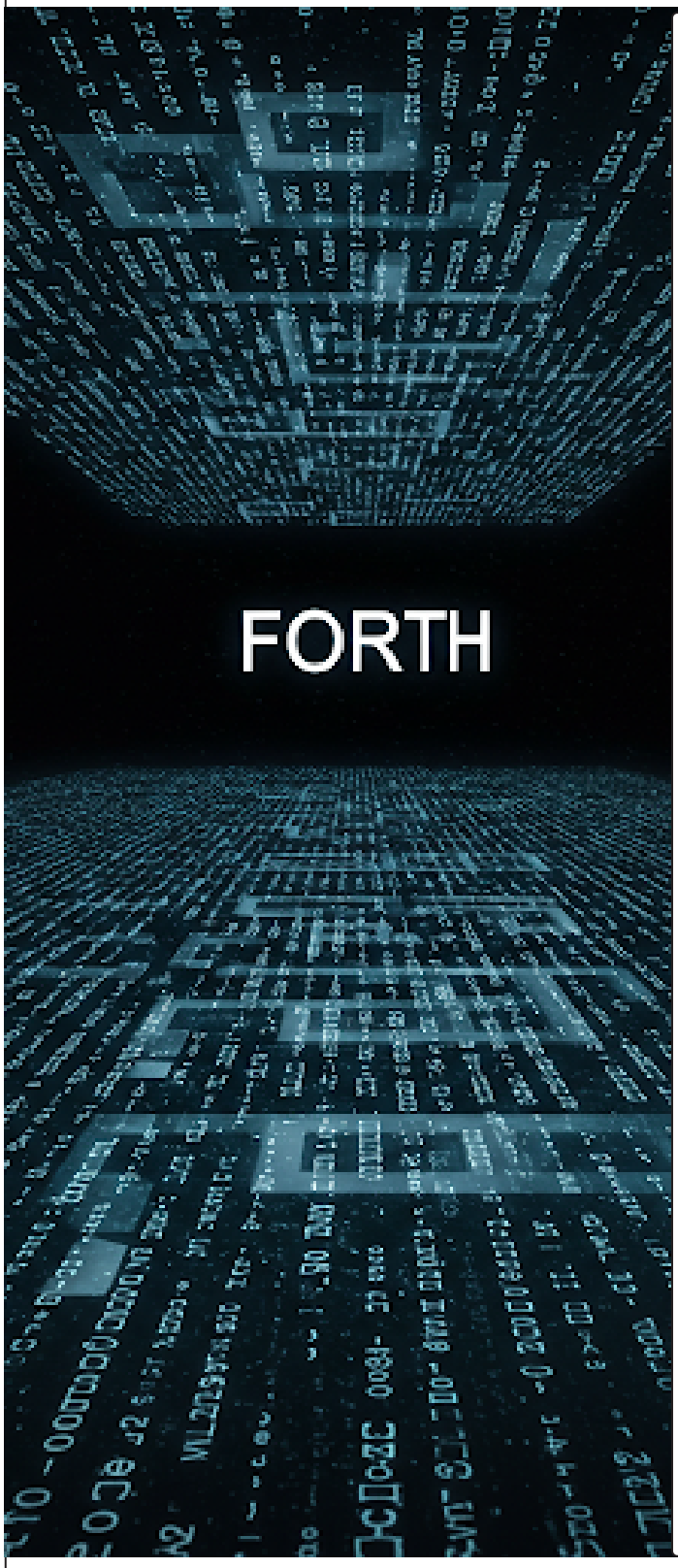
Vierte Dimension

1/2025

Das Forth–Magazin

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

In dieser Ausgabe:



Euroforth 2024

Statische Typprüfung und Objektorientierung

Ein Schachspiel, gemeinsam mit Google Gemini KI erstellt

Breadboards III

About NEED and noForth t

Leserbriefe und Meldungen 5

Euroforth 2024 9
Anton Ertl

Statische Typprüfung und Objektorientierung 11
Stephan Becher

Ein Schachspiel, gemeinsam mit Google Gemini KI erstellt 16
Daniel Ciesinger, Cornu GmbH

Breadboards III 26
Rafael Deliano

About NEED and noForth t 29
Albert Nijhof

Titelbild: Zukunft der Programmiersprache Forth AUTOR: Gemini

Quelle: Eigenes Werk (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

Liebe Leser,

zum Jahresanfang kommt hier nochmal eine etwas dickere Ausgabe unseres Forth–Magazins. Darüber hinaus gibt es sogar bereits Material für das nächste Heft. Ich weiß gar nicht, wann in der 40–jährigen Geschichte unseres Magazins das schon mal so war und bin stolz auf euch! Meinen herzlichen Dank an alle Autoren!

Ihr habt es schon gesehen, das Titelbild wurde KI–unterstützt gemalt. Wer ist dann eigentlich der Künstler, die KI oder der Schreiber des Prompts? Oder der Bearbeiter des Bildes? Bei wem liegen die Rechte am Bild eigentlich? Ich fürchte, da kommt noch Seltsames auf uns zu.

Die *NEED CHALLENGE* ist gestartet und Albert Nijhof hat die Herausforderung angenommen, knapp aber treffend zu zeigen, wie `NEED` gedacht ist und wie er es im noForth t bereits verwirklichen konnte. Das ist wirklich ein interessanter Ansatz, den er da hat. Die *LIBRARY* mit ihren *CHAPTERs* gleich ins üppige Flash des Micros zu packen und von dort die Forth–Applikation interaktiv im RAM zu entwickeln. Ich hoffe sehr, weitere Forth–Entwickler folgen dem Beispiel, zeigen, wie sie es gemacht haben und auf welcher Plattform. Und `NEED` zieht Kreise bis hin zu den einzelnen Forth–Benutzern landauf, landab.

Anton Ertl war in *Newcastle* bei der EuroForth 2024 und berichtet euch von seinen Eindrücken und dem Tagungsprogramm dort. Damit ihr, die ihr daheim geblieben seid, die Reise zumindest in Gedanken nachvollziehen könnt. Und wisst, wo ihr nun die Forth–Beiträge von der Tagung finden könnt. Ich finde es sehr schön, dass Reisen wieder möglich ist und hoffe, ihr macht auch alle tüchtig Gebrauch davon. Die Forth–Tagung kommt ja auch schon bald.

Stephan Becher verfolgt mit *StrongForth 3.0* sein Ziel weiter, unter Windows ein solides Forth–System zu etablieren. Und dringt damit nun auch in den Bereich der Objektorientierung vor — spannend.

Daniel Ciesinger hat mich dann mit richtig viel Forth–Code und einem Spiel überrascht: *Schach*. Das Besondere: Er hat dafür verschiedene KI–Systeme auf Ihre Tauglichkeit für Forth getestet. Was soll ich dazu sagen? Das werden ja tolle Zeiten für Forth.

Und Rafael Deliano setzt die Serie weiter fort, in der er Einblick ins praktische Arbeiten mit der Hardware gibt, in der euer Forth ja was tun soll. Techniken für den Anbau *analoger Komponenten* werden diesmal behandelt — vielleicht schickt der eine oder andere mir ja mal *seine eigene* Vorgehensweise.

Bernd Paysan wird im Frühjahr 2025 eine *leibhaftige* Forthtagung organisieren! Auf der Rückseite dieses Heftes findet ihr alles zu Ort und Zeit und Anmeldung — sei schnell, sei dabei!

Haltet auch Ausschau danach auf unserer Webseite `www.forth-ev.de` — bis bald dort!

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-2025-01`

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

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Gerald Wodni

Maker Faire

Messen sind der beste Weg, das Publikum zu erreichen. Auch für Vereine. Die Maker formierten sich zwar erst nach 2005 in den USA. Die Entwicklung, oder besser gesagt Kommerzialisierung, erfolgte aber rasch. Die erste „Maker Faire“ 2006 dort hatte bereits 20k Besucher.

Der Name ist ein geschütztes Warenzeichen, das Geschäftsmodell wird lizensiert. In Deutschland an den *Heise–Verlag*. Der dann die hiesigen Veranstalter kostenpflichtig unterstützt. Wie bei jedem Franchise überwiegen die Vorteile gegenüber diesen Kosten. Unabhängige Veranstalter wie „Make Munich“ florieren meist nur in guten Zeiten.

In Deutschland erfolgte der Start verzögert ab 2013 (Abb. 2), die Veranstaltungen entwickelten sich aber recht gut. Bis 2020 per Virus eine harte Landung erfolgte, die zu einer sofortigen Marktbereinigung führte. Allerdings war der Boom wohl ohnehin vorbei. Mit Blick auf die vergleichbaren Modellbau–Messen ist anzunehmen, dass sich etwa drei größere Veranstaltungen halten können. Und regional wohl noch mal die gleiche Zahl kleiner Messen, die aber kommen und gehen.

Für Aussteller, die nichts verkaufen, fallen keine Standgebühren an [1].

Wer sich schlecht präsentiert, wird vom Publikum ignoriert. Ein Poster als Blickfang sollte sein. Stellen wir uns vor, wir nehmen den berühmten Apple–Werbespot von 1984 und machen etwas daraus (Abb. 1), Format mindestens A2. Für standfeste Mechanik auf dem Tisch ist man selber zuständig. Geeignet sind Notenständer und Magnethalter für Poster.

Stapel alter VDs zum Verteilen bieten sich an, da einzige nennenswerte Aktivität des Vereins. Vorzugsweise aber nicht steinalte Jahrgänge. Und solche, bei denen das Cover etwas ansehnlicher ist. Viele Besucher sammeln heutzutage aber kein Papier mehr. Visitenkarten mit Link auf Webseite als 2D–Code empfiehlt sich.

[1] `https://maker-faire.de/aussteller-faq/`

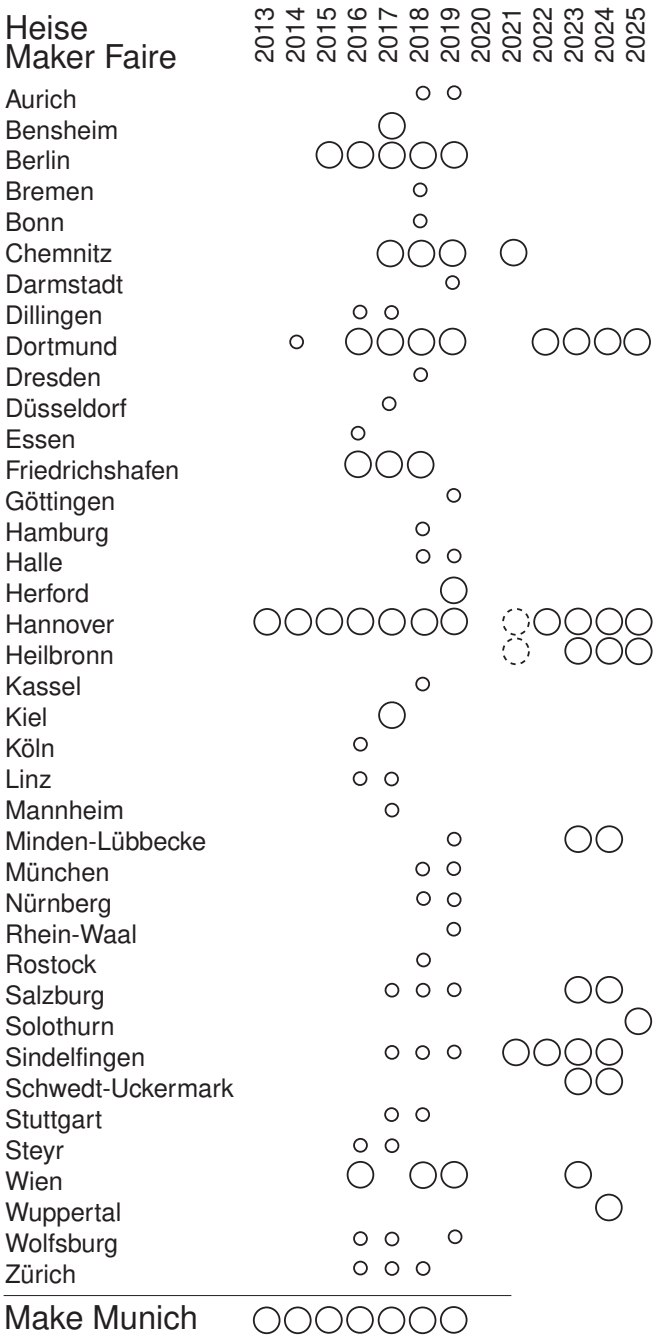


Abbildung 2: Historie der Maker Fair

Abbildung 1: Postervorschlag

Die Lisp–Box

„Retro–Computing–Feeling, aber mit Zukunftssicherheit und hohem Nutzwert — das bietet die Lisp–Box, ein schneller Bastel–Heimcomputer ohne Betriebssystem … Ein Rechner, den man bis hinab zu den letzten Bits und Registern programmieren, erforschen und verstehen kann. So stellen sich viele, die wie ich im *Goldenen Zeitalter der Heimcomputer* aufgewachsen sind, noch heute den idealen PC vor — kein Betriebssystem installieren, keine Blackbox unter der Haube, direkt nach dem Einschalten loslegen wie damals … Eigentlich wurde ich schon vor einigen Jahren auf die Programmiersprache uList aufmerksam, die wie MicroPython, CircuitPython oder TinyBasic die

> Programmierung und Steuerung vom Mikrocontrollern ohne ständiges Rekompilieren verspricht . . .“ [1]

Verwendet wird *uLisp*. Hardware ist ein Arduino mit Tastatur und Display, in Schachtel verpackt. Die Darstellung ist auf 8 Seiten recht ausführlich.

Nachdem die 3D–Drucker–Welle ausgeklungen ist, sortiert sich bei den Makern Gängiges wie Elektronik ein. Aber auch Exotisches wie längst vergangene Altertümer haben den Reiz des Neuen.

Sicher auch Chancen für Forth, wollte man sie denn nutzen. JRD

[1] Hartmut Grawe „Die Lisp–Box“ Make: 2/2024
[2] `www.heise.de/ratgeber/Schritt-fuer-Schritt-Einen-uLisp-basierten-Computer-bauen-10170625.html`

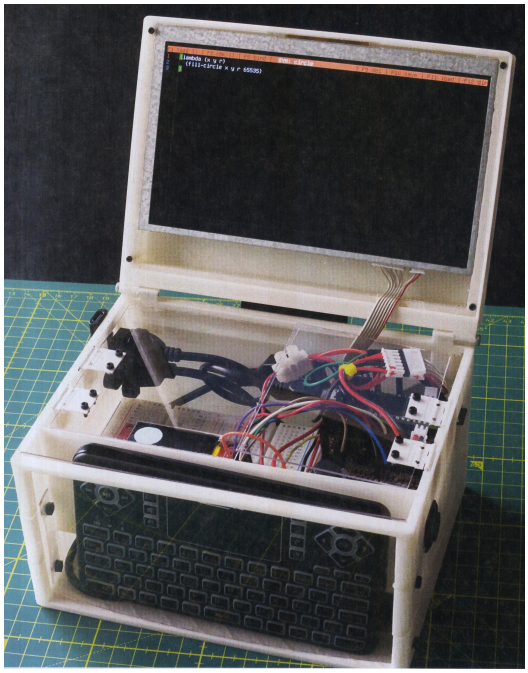


Abbildung 3: Die Lisp–Box

Ein 10b Instructions Forth OS in 46 Bytes

Auf *sourcehut* hat Philippe Brochard jüngst sein 10bi–ForthOS vorgestellt, „a full 8086 OS in 46 bytes“.

> “10biForthOS = 10b(2) i(nstructions) Forth OS is a very primitive Forth with only two instructions:
> 1 is compile
> 0 is execute
>
> It is heavily inspired by Frank Sergeant’s 3–Instruction Forth . . .
>
> When loaded after the boot, 10biForthOS listens on the serial port for instructions. The 1 instuction should be followed by an assembly opcode to be compiled into a fixed memory location. The 0 instruction launches the compiled program.
>
> And that’s it!
>
> On a host computer you can then send commands to 10biForthOS. As examples a subleq–eForth or even sectorc can be used to code in eForth or C . . .”

Auf seiner Seite gibt Phillipe einige hübsche Code–Beispiele an, die zeigen, wie dieses winzige Betriebssystem arbeitet. Unter anderem, wie man damit ein eForth hochzieht — erstaunlich. Und natürlich den Quellcode.

Auch geht er der Frage nach, ob das schon ein Forth sei.

> “It seems to be: it has an outer interpreter which understands assembly opcodes and an inner interpreter: the standard machine code. You can load/redefine code at will.
>
> Even if it lacks stacks, dictionnary, defining words, etc. It has the simplicity and hacky feeling of Forth.”

Vielen Dank, Carsten, für den Hinweis hierauf.[1] mka

`https://git.sr.ht/~hocwp/10biForthOS#10biforthos-a-full-8086-os-in-46-bytes`

Die Top–Programmiersprachen 2024

Alle Jahre wieder wird vom *IEEE Spectrum* der Datensalat durchgekaut, um auf mehr oder weniger sinnvolle Schlussfolgerungen zu kommen. Ausgewertet wurden Daten zu 63 Programmiersprachen. Der *allgemeine Spectrum–Index* soll die Verwendung durch IEEE–Mitglieder und Programmierer abbilden. Der *Jobs–Index* speist sich aus Stellenanzeigen. Das *Trending* zeigt die dynamische Veränderung. Die Zeiten ändern sich eben.

C hat es inzwischen auch erwischt:

> “C’s popularity appears to be on the wane, falling from fourth to ninth place on the Spectrum ranking and from 7th to 13th on the Jobs ranking.”

Und *Forth* ist inzwischen rausgefallen wegen Bedeutungslosigkeit:

> “This year also saw several languages drop out of the rankings. This doesn’t mean a language is completely dead, it just means that these languages’ signal is too weak to allow them to be meaningfully ranked. Languages that dropped out included Forth, a personal favorite of mine that’s still popular with folks building 8–bit retro systems because of its tiny footprint.”

2017 landete Forth im Spectrum–Index noch auf Platz 48.

Den Jobs–Index führt immer noch *SQL* an, dicht gefolgt von *Python* und *Java*.

[1] Und, lieber Leser, schick gerne her, was dir da draußen so begegnet.

Im Trend liegt *Python* mit großem Abstand vor allen anderen. JRD

`https://spectrum.ieee.org/top-programming-languages-2024`

`https://spectrum.ieee.org/top-programming-languages-methodology-2024`

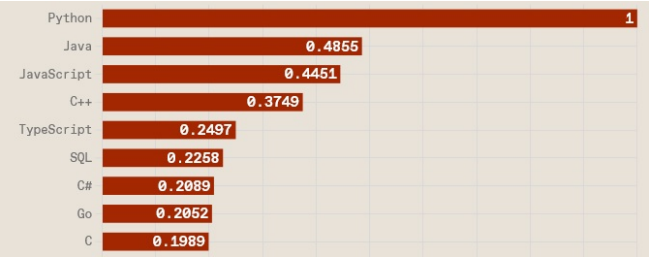


Abbildung 4: Die oberen 9 von 55 gelisteten (und genutzten?) Programiersprachen

Analogtechnik? Williams lesen!

Die Analog–Technik ist ganz nah an Forth gerückt. Mit dem Einsatz von Sensoren und Übermittlung der Daten per Funk bewirken das die neuen ESPs. Sei es zusammen mit noForth auf MSP430, RISC–V oder RP2040, sei es als ESP32. Das zu testen und sauber auszulegen, ist nicht trivial. Im Heft lest ihr davon beim Breadboard–Layout. Dort erwähnt der Autor auch seine Quelle: Williams. Einige von euch kennen ihn womöglich schon von Anfang an. Den anderen sei er hier kurz vorgestellt: Wer war dieser Analog–Guru?

> „James M. Williams (14. April 1948 – 12. Juni 2011) war ein US–amerikanischer Pionier bei der Entwicklung *analoger* integrierter Schaltkreise und Autor zu Themen der Analogtechnik. Er arbeitete für das Massachusetts Institute of Technology (1968–1979), National Semiconductor (1979–1982) und die Linear Technology Corporation (LTC) (1982–2011). Williams schrieb über 350 Publikationen im Zusammenhang mit dem Entwurf analoger Schaltungen, darunter fünf Bücher, 21 Applikationsschriften für National Semiconductor, 62 Applikationsschriften für Linear Technology und über 125 Artikel für das EDN[2]–Magazin.
>
> Er wurde 1992 mit dem ‚Innovator of the Year Award' des EDN–Magazins ausgezeichnet und wurde 2002 in die ‚Electronic Design Hall of Fame' gewählt.
>
> Williams erlitt am 10. Juni einen Schlaganfall und starb am 12. Juni 2011." [Wikipedia]

Tragisch dabei: Bob Pease, ein weiterer Pionier der Analogtechnik, starb auf dem Rückweg von Jim Williams' Beerdigung bei einem Verkehrsunfall.

`https://www.edn.com/analog-guru-jim-williams-dies-after-stroke/`

Abbildung 5: Jim Williams' Thermometer Skulptur

Autorenverzeichnis aus 40 Jahren Forth–Magazin „Vierte Dimension" (4d)

2024 wurde die 4d *40 Jahre* alt. Das habe ich zum Anlass genommen, meinen Sohn, Christopher Kalus, zu bitten, die Namen aller Autoren aus den Inhaltsverzeichnissen des 4d–Archives zu extrahieren, was er dankenswerterweise mit einem Python–Programm[3] gemacht hat. Und so gibt es im 4d–Archiv nun auch das *alphabetische Autorenverzeichnis*.

`https://wiki.forth-ev.de/doku.php/vd-archiv`

Und wir haben, neugierig wie wir sind, dabei auch mal ausgewertet, wie viele Beiträge ein Autor über die Jahre eigentlich eingereicht hat.

153 mal blieb es eine einmalige Sache. Wobei es sicherlich interessant wäre, mal nachzuforschen, was deren Motiv war, zu schreiben und was sie heute so machen.

Dann gibt es eine größere Gruppe von 49 Namen, die 2 bis 3 mal etwas beigetragen haben und eine fast ebenso große Gruppe, die 4 bis 9 mal geschrieben haben.

Und dann kommen wir zu den Vielschreibern. So nenne ich mal die Gruppe der Autoren, die 10 mal und öfter Beiträge verfasst haben.

Ich habe lange mit mir gerungen, ob die gelistet werden sollten. Einerseits gebührt ihnen unser aller Anerkennung für ihre Leistung über all die Jahre. Andererseits sollen andere Autoren davon natürlich nicht abgeschreckt werden: Jeder Beitrag zählt, auch die einmaligen!

Nun denn, hier ist die Liste:

1. Behringer Fred : 234
2. Deliano Rafael : 98
3. Prinz Friederich : 92
4. Kalus Michael : 70

[2] EDN (Electronic Design News); seit 1956; bis Juni 2013 monatlich in gedruckter Form herausgegeben, danach als Online–Publikation.
[3] Listing auf Wunsch beim Author.

5. Paysan Bernd : 62
6. Bitter Martin : 56
7. Hoffmann Ulrich : 52
8. Vinerts Henry : 39
9. Klingelnberg A. : 38
10. Strotmann Carsten : 33
11. Staben Jörg : 32
12. Wälde Erich : 29
13. Koch Matthias : 25
14. Vogt Claus : 24
15. Plewe Jörg : 24
16. Trute Matthias : 22
17. Ertl Anton : 16
18. Schleisiek Klaus : 15
19. Teich Johannes : 14
20. Ouwerkerk Willem : 11
21. Krinninger Ch. : 11
22. Kohl Klaus : 11
23. Mahlow Manfred : 10
24. Minke Ron : 10
25. Schemmert Wolfgang : 10
26. Schleisiek K.–P. : 10
27. Tiedemann S. : 10

Dabei wurde nicht berücksichtigt, wann diese Beiträge gewesen sind. Es gab Cluster. Einige Autoren sind eine Zeit lang einem bestimmten Motiv gefolgt und gaben eine gewisse Serie von Beiträgen dazu ab. Andere haben kontinuierlicher immer mal wieder geschrieben.

Und einer hatte eine eigene Kolumne: Fred Behringer. Er hat lange Zeit „von über'm großen Teich" berichtet, was sich in der Forth–Szene da drüben so tat, gewonnen aus seiner Korrespondenz. Bei Friederich Prinz war es ähnlich. Und ich denke, ihre Beitragszahlen wären noch größer, lebten sie noch.

Doch es gibt einen, der immer weiter schreibt und damit nun der Autor mit den meisten Einzelbeiträgen im Forth–Magazin geworden ist. Ein Mann, der Forth auch beruflich schon seit langer Zeit nutzt, vor allem im Bereich des *Embedded Computing*. Und das ist Rafael Deliano.

Über die Zeit kamen immer wieder Beiträge von ihm herein. Was mich als Editor vieler Hefte über die Jahre tatsächlich auch immer wieder motiviert hat, damit weiterzumachen.

Also, Rafael, von dieser Stelle daher zu dem Dank aller Leser auch meinen ganz persönlichen herzlichen Dank dafür!

mk

Die kommende VCF in München

„Zum vierundzwanzigsten Mal kommt das VCF am Wochenende vom *3./4. Mai 2025* in unser schönes München.

Ziel des Vintage Computer Festivals ist es, den Erhalt und die Pflege ‚historischer' Computer und anderer (E)DV Gerätschaften zu fördern, das Interesse in ‚überflüssiger' Hard– und Software zu wecken und vor allem den Spaß daran auszuleben.

Das VCF ist nicht nur im kalifornischen Silicon Valley ein regelmäßiges Ereignis, sondern inzwischen auch ein fixer Punkt in den europäischen Terminplänen.

Also lasst uns zurückkehren in die *Guten Alten Tage*, als Hacker noch keine Sicherheitsberater, Bytes noch keine Megabytes und *Kleine Grüne Männchen* noch *Kleine Gruene Maennchen* waren!"

`https://www.vcfe.org/D/`

VCFe 23.0 auf YouTube

Neulich ist dort ein Videoclip erschienen, in dem Hans euch durch die VCFe 23.0 vom letzten Jahr führt und die wichtigsten Exponate und Installationen präsentiert und die Atmosphäre dort gut wiedergibt — als sei man dabei gewesen.

Gleich zu Begin seht ihr Carsten Strotmann, der 8"–Floppy–Laufwerke (!) am Atari 130XE demonstriert. Und um 07:16 im Clip taucht sogar unsere VD auf bei einem 6809–Selbstbaurechner, auf dem das Z79Forth von Francois Laagel läuft. Rafael Deliano hat uns dort entdeckt, als er sich für den Lochstreifenleser interessierte, der in dem Clip gezeigt wird, da er selbst grad auch einen solchen in Bau hat. Das Ding lief dort am MOS TIM–1 Bausatz–Computer[4] von 1974.

Bin gespannt, wie Forth darin eingesetzt werden wird.

`https://www.youtube.com/watch?v=IAC49NU_meg`

[4] Sein „Pfleger" wurde leider nicht namentlich genannt

Euroforth 2024

Anton Ertl

Die EuroForth 2024 wurde von Bill Stoddart mit Unterstützung von Janet Nelson organisiert.

Touristisches

Newcastle liegt im Nordosten Englands am Fluss *Tyne*. Die Stadt hatte schon im 16. Jahrhundert ein florierendes Geschäft mit Kohle. Besonders in der industriellen Revolution wuchs und gedieh die Stadt und ihre Umgebung, vor allem entlang des Tyne. So entstand ein prächtiges Stadtzentrum im neoklassischen Stil — Grainger Town.

Außerhalb des Zentrums sind die Bauten deutlich moderner und der Verkehr ist sehr autozentriert organisiert, mit Radstreifen zum Marginalisieren des Radverkehrs und mit Bettelampeln gegen Fußgänger, mit so langen Rotphasen, dass die Fußgänger in den meisten Fällen dann bei Rot drübergehen; dafür sind die Grünphasen um so kürzer. Diese Bettelampeln gibt es auch im Zentrum, aber in der Fußgängerzone ist man nicht alle paar Meter mit einer konfrontiert. Weiters konnte ich entlang des Tyne ohne Bettelampeln laufen, und zum Großteil, ohne um Privatgrundstücke herumlaufen zu müssen, anders als an manch anderen Ufern.

Unser Hotel war knapp außerhalb des Zentrums in der Nähe der *Gateshead Millenium Bridge*, die wir jedesmal überquert haben, wenn wir zu einem Restaurant gingen (Abb. 1).

Die 2001 eröffnete Gateshead Millenium Bridge ist eine Brücke für Fußgänger und Radfahrer, die gekippt werden kann, um Schiffe durchzulassen (Abb. 2). Leider konnten wir das während unseres Aufenthalts nicht erleben.

Abbildung 1: Millenium Bridge, geschlossen ... [JimmyGuano (CC BY–SA 4.0)]

Abbildung 2: ... und geöffnet. [en.wikipedia–User Mike1024]

Im Zentrum war am Wochenende viel los; einerseits gab es ein Fußballspiel Newcastle gegen Manchester City (unentschieden), es gab Drachenbootrennen auf dem Tyne, und viele der 60.000 Studierenden der beiden Universitäten in Newcastle waren offenbar in der Innenstadt unterwegs, die geradezu überging vor jungen Leuten.

Abbildung 3: Die Charles Gray Säule. [Nigel Thompson (CC BY–SA 2.0)]

Ein Sohn der Stadt, eigentlich der Umgebung, ist Charles Gray (1764–1845), der zweite Earl Grey[1] [2], der eine lange und sehr erfolgreiche politische Karriere hinlegte, vom Abgeordneten im Parlament ab 1786 bis zum Premierminister des britischen Empire 1830–1834. In

[1] Der mit Bergamottöl verhunzte Tee ist nach ihm benannt, aber es gibt keinen Beleg, dass der Tee, abgesehen vom Namen, irgendetwas mit ihm zu tun hat.

[2] Citrus × Limone, Syn.: Citrus bergamia; Die Frucht wird hauptsächlich wegen der ätherischen Öle angebaut, die in der Bergamottenschale enthalten sind. Bergamottöl spielt vor allem in der Parfümindustrie eine Rolle, wird aber auch zum Aromatisieren von Tees (Earl Grey) verwendet. [wikipedia]

seiner Amtszeit als Premierminister wurde 1832 die Wahl zum Parlament — für England und Wales — reformiert und im Ergebnis demokratischer und 1833 die *Sklaverei* fast im ganzen Empire abgeschafft. Ein Denkmal für Charles Gray steht auf einer fast 40 m hohen Säule an prominenter Stelle in Newcastle (Abb. 3), und die Gray Street ist wohl *die* Prachtstraße Newcastles.

Robert Stephenson wurde 1803 in der Nähe geboren, und hat seine Lokomotive *Rocket* erstmals 1829 in den *Forth Street Works* seiner Firma in Newcastle gebaut.[3]

Das *Social Event* am Samstag war eine Stadtbesichtigung inklusive Besuch einer Gallerie. In der *Forth Street* waren wir dabei aber nicht. Das Abendessen am Samstag hatten wir im *Blackfriars*, einem Restauraunt im Refektorium aus dem 13. Jahrhundert eines ehemaligen Dominikanerklosters (Abb. 4).

Abbildung 4: Priorei Blackfriars. [en.wikipedia–User Dposte46]

Vorträge

Anton Ertl In *How to Implement Words (Efficiently)* erzählte ich, welche Implementierungstechniken es für die Ausführung von Wörtern — insbesondere solchen, die mit `create ... does` definiert wurden, aber auch anderen — in verschiedenen Arten von Systemen gibt. Auf manchen entstehen *hohe Kosten* durch nebeneinanderliegenden Code und Daten (100 Zyklen und mehr), und durch falsche Sprungvorhersage bei returns (20 Zyklen und mehr). Eine Reihe von Microbenchmarks zeigt diese Achillesfersen bei verschiedenen Forth–Systemen auf. Aber man kann Wörter auch ohne diese Achillesfersen implementieren.

Nick Nelson sprach in *A Forth binding for GTK4* über die Schwierigkeiten, die die Umstellung des Grafik–Toolkits von GTK3 auf GTK4 bietet, und wie ein neuer Ansatz für die Bindung auf der Grundlage der einzigartigen Features von Forth es erlaubt hat, das Problem zu lösen.

François Laagel erzählte uns davon, welche Features das Text–Terminal *VT420* bietet, u. a. benutzerdefinierte Fonts (seit der VT200–Serie), und wie er *Pac–Man* mit einem Display auf einem VT420 implementiert hat, und was dazu alles nötig war.

Bill Stoddart setzte in *Prospective Values and Forth* seine Arbeit zum reversiblen Forth (mit Backtracking) fort.

Nick Nelson präsentierte uns *Bekki*, das Forth–Gegenstück zu Alexa, Siri und Co. Insbesondere spricht Forth jetzt mit uns und kann auch Sprachkommandos entgegennehmen. Das ist besonders relevant für den Einsatz in industriellen Umgebungen, wo man nicht unbedingt eine Hand frei hat. Allerdings ist Bekki derzeit eine Machbarkeitsstudie, kein fertiges Produkt.

Anton Ertl In *The Performance Effects of Virtual–Machine Instruction Pointer Updates* präsentierte ich den Abschlussvortrag zu einer Arbeit, von deren ersten paar Schritten ich schon 2023 berichtet hatte: Wie man Instruction–Pointer–Updates in Gforth optimieren kann, und wie viel das bringt. Auf manchen Benchmarks bringt es mehr als einen Faktor 2 auf modernen Prozessoren. Im Vergleich zum Vorjahr ist die Optimierung von Sprüngen auf VM–Ebene dazugekommen und die Präsentation wurde vollkommen überarbeitet.

Gerald Wodni präsentierte *Warpgate*, das C–Interface für mehrere Forth–Systeme: Gforth, VFX, und hoffentlich auch SwiftForth, und im Prinzip für alle, die es implementieren wollen. Wobei der Großteil der Arbeit von SWIG (einem C–Compiler, spezialisiert auf Interfaces zwischen C und anderen Sprachen) übernommen wird, noch mehr als bei seinen früheren Ansätzen ohne und mit SWIG.

Paul Bennet stellte sein *Projekt eines Buchs über Forth* für Kinder ab 5 Jahren vor.

Bill Stoddart erzählte davon, wie er Schwierigkeiten mit der Weiterentwicklung seines *reversiblen Forth–Systems* (auf 32–Bit–Basis) dadurch umging, dass er in einer virtuellen 32–Bit–Maschine entwickelt.

Anton Ertl Bericht über die *Fortschritte im Gforth–Decompiler;* aber nichts, was ich nicht schon in meinem Vortrag zu dem Thema auf der Forth–Tagung 2023 erzählt habe.

Euroforth Proceedings

`http://www.euroforth.org/ef24/papers/`

`http://www.euroforth.org/ef23/papers/`

[3] Mit seiner Lokomotive *The Rocket* gewann Robert Stephenson das legendäre Rennen von *Rainhill*. Nach diesem Erfolg lieferte das Unternehmen weitere Lokomotiven für die Liverpool and Manchester Railway und andere Bahnlinien. [wikipedia]

Statische Typprüfung und Objektorientierung

Stephan Becher

Ein Forth–System mit statischer Typprüfung kann grundsätzlich kein Standardsystem im Sinne der Forth–2012–Spezifikation sein. Denn statische Typprüfung zur Kompilierzeit ist nur dann möglich, wenn im Quellcode Wörter mit Stackdiagrammen versehen sind. Im Standard sind Stackkommentare hingegen kein Muss. Außerdem können in einem System mit statischer Typprüfung keine Wörter implementiert werden, deren Stackeffekte zur Kompilierzeit nicht eindeutig sind. `?DUP PICK ROLL N>R` *und* `NR>` *sind Beispiele dafür.*

Warum sollte also nicht auch in anderen Punkten vom Standard abgewichen werden? Einige historisch gewachsene Techniken sind umständlich und wenig zeitgemäß. Zudem stehen sie oft der Erzeugung schnellen Codes im Wege. Das Ziel sollte sein, Forth zu modernisieren und fortgeschrittene Programmiertechniken zu ermöglichen.

Eines der wichtigsten Schlüsselworte der vergangenen Jahrzehnte im Bereich Programmiersprachen ist Objektorientierung. Die überragende Flexibilität der Programmiersprache Forth hat zwar die Implementierung von Bibliotheken ermöglicht, die objektorientierte Programmierung unterstützen, doch im Standard ist davon nichts zu finden. Der Grund mag darin liegen, dass jede dieser Bibliotheken eigene Wege geht und sich bisher keine davon allgemein durchsetzen konnte. Objektorientierte Programmierung in Forth dürfte zudem erst dann Verbreitung finden, wenn auch der Kern des Systems objektorientiert ist. Wenn also Wörter, Wortlisten, Input– und Output–Streams, Speicherbereiche und andere Systemstrukturen Objekte von Klassen sind.

Diese Konzepte einzuführen, erfordert ein grundlegendes Update von Forth, eine Art Forth++. Es liegt daher nahe, in einem System mit statischer Typprüfung zugleich objektorientierte Konzepte einzuführen. Das im Folgenden vorgestellte Konzept basiert in großen Teilen auf dem von C++ und fügt sich nahtlos in ein Forth–System mit statischer Typprüfung ein.

Datenfelder und Konstruktoren

Hier ist ein einfaches Beispiel einer Klassendefinition in einem objektorientierten Forth und wie sie benutzt werden kann:

```
dt object procreates rectangle
class rectangle
  null signed member px
  null signed member py
  null unsigned member width
  null unsigned member height
  : rectangle ( rectangle -- 1st )
    dup erase ;
endclass
new rectangle constant rect1
+100 rect1 px !
+150 rect1 py !
40 rect1 width !
25 rect1 height !
```

Der Datentyp `object` ist eine Klasse, die als direkter oder indirekter Vorfahre aller anderen Klassen dient. Im Beispiel wird zunächst mit `procreates` ein neuer Datentyp `rectangle` von `object` abgeleitet. Es folgt die Klassendefinition. Sie besteht aus vier Datenfeldern, die die Koordinaten der unteren linken Ecke eines Rechtecks sowie dessen Breite und Höhe beschreiben. `member` ist ein Definitionswort, das ähnlich wie `constant` einen Wert eines beliebigen Datentyps als Parameter erwartet, damit dem erzeugten Datenfeld dieser Datentyp zugeordnet werden kann. Weil die Datenfelder nicht unmittelbar initialisiert werden, spielt der konkrete Wert keine Rolle.

Das Stackdiagram von `px` lautet beispielsweise

```
( rectangle -- address -> signed )
```

`px` erwartet somit ein Objekt des Datentyps `rectangle` auf dem Stack und liefert die Adresse des zugehörigen Datenfelds zurück.

```
: rectangle ( rectangle -- 1st )
  dup erase ;
```

definiert den Konstruktor der Klasse `rectangle` , der den gleichen Namen wie die Klasse trägt. Er initialisiert neu erzeugte Objekte, indem er alle Datenfelder mit dem Wert null füllt. Das erledigt eine überladene Version des Wortes `erase` , die direkt auf beliebige Objekte angewendet werden kann:

```
erase ( object -- )
```

Eine Klasse kann mehrere überladene Konstruktoren besitzen. Man könnte z. B. eine zweite Version des Konstruktors hinzufügen, die Initialisierungswerte für die vier Datenfelder auf dem Stack erwartet. Das Stackdiagramm eines Konstruktors muss immer ein Objekt seiner Klasse als letzten Eingangsparameter und als einzigen Ausgangsparameter haben.

```
new rectangle constant rect1
```

erzeugt ein Objekt der Klasse `rectangle` im dynamischen Speicher und führt anschließend den Konstruktor aus. Das funktioniert auch, wenn `new` kompiliert wird. Weil der Konstruktor das neue Objekt zurückliefert, kann es in diesem Beispiel sofort als Konstante definiert werden.

Auf die Datenfelder kann nun wie auf Variablen zugegriffen werden, wobei jedoch immer das konkrete Objekt

angegeben werden muss. Die Namen der Datenfelder dürfen in anderen Objekten wiederverwendet werden. Namenskonflikte sind ausgeschlossen, denn Interpreter und Compiler können anhand des Datentyps des Parameters erkennen, welche Klasse gemeint ist.

Methoden

Wörter, die innerhalb der Klassendefinition definiert werden und sich jeweils auf ein bestimmtes Objekt der Klasse beziehen, heißen *Methoden*. Der Konstruktor ist solch eine Methode. Für die Klasse `rectangle` könnte innerhalb der Klassendefinition beipielsweise eine zusätzliche Methode `move` definiert werden:

```
: move ( signed signed rectangle -- )
  tuck py +! px +! ;
```

Das Forth–2012–Wort `move` kann ohne Weiteres überladen werden, weil sich die unterschiedlichen Versionen anhand der Datentypen der Parameter unterscheiden lassen. Da sich Methoden immer auf die Objekte ihrer Klasse beziehen, haben sie in der Regel ein Objekt als letzten Eingangsparameter.

Hätte die Klasse `rectangle` keinen Konstruktor, könnte `new` keine Objekte erzeugen. Das kann sinnvoll sein, wenn eine Klasse lediglich als Elternklasse für andere Klassen dienen soll. Ein Beispiel ist die Klasse `object` :

```
dt single procreates object
class object
  virtual delete ( object -- )
  :noname ( object -- )
    cast address free ; is delete
endclass
```

Diese Klasse hat weder Datenfelder noch einen Konstruktor. Sie verfügt lediglich über die virtuelle Methode `delete` , mit der der für das jeweilige Objekt reservierte dynamische Speicher freigegeben wird. Weil Methoden ebenso wie Datenfelder vererbt werden, steht `delete` auch in der Klasse `rectangle` zur Verfügung. Komplexere Klassen erfordern möglicherweise mehr oder weniger umfangreiche Aufräumarbeiten, bevor sie endgültig gelöscht werden dürfen. Beispielsweise muss ein File geschlossen werden, das der Konstruktor geöffnet hat, oder der Speicherplatz eines vom Konstruktor angelegten Puffers muss freigegeben werden. In solchen Fällen muss eine klassenspezifische Version der virtuellen Methode `delete` bereitgestellt werden.

Was ist nun der Unterschied zwischen einer normalen Methode wie `move` und einer virtuellen Methode? Eine normale Methode wird bereits zur Kompilierzeit an das jeweilige Objekt gebunden. Der Compiler legt also wie bei jedem anderen Wort fest, welche Methode zur Laufzeit ausgeführt wird. Bei virtuellen Methoden wird diese Entscheidung erst zur Laufzeit getroffen. Man spricht von *late binding*. Hätte die Klasse `rectangle` eine eigene Instanz der virtuellen Methode `delete` , würde diese Instanz in folgendem Beispiel aufgerufen:

```
... rect1 cast object delete ...
```

Wäre `delete` hingegen keine virtuelle Methode, würden sowohl der Interpreter als auch der Compiler hier die entsprechende Instanz der Klasse `object` aufrufen.

Vererbung

Vererbung ist eines der zentralen Konzepte objektorientierter Programmierung. Es ist in einem System mit statischer Typprüfung bereits für Datentypen realisiert und kann nahezu unverändert für Klassen übernommen werden. Beispielsweise erben alle vom Datentyp `single` abgeleiteten Datentypen, etwa `integer` und `address` , die Funktionalität der Wörter `dup` und `drop` . Der vom Datentyp `double` abgeleitete Datentyp `integer-double` erbt die entsprechenden überladenen Versionen von `dup` und `drop` . Jede Klasse erbt wiederum die Datenfelder und Methoden, einschließlich der virtuellen Methoden, aller seiner Vorfahren. Eine Ausnahme ist der Konstruktor. Da der Konstruktor den Namen der Klasse trägt, muss jede Klasse, von der Objekte erzeugt werden sollen, mindestens einen individuellen Konstruktor besitzen, auch wenn dieser Konstruktor lediglich den Konstruktor seines direkten Vorfahrens aufruft.

Von der Klasse `rectangle` ließe sich beispielsweise eine weitere Klasse ableiten:

```
dt single procreates color
dt rectangle procreates colored-rectangle
class colored-rectangle
  null color member my-color
  : colored-rectangle
    ( colored-rectangle -- 1st )
    rectangle ;
  : colored-rectangle
    ( color colored-rectangle -- 2nd )
    rectangle tuck my-color ! ;
endclass
```

Die Klasse `colored-rectangle` verfügt neben den von `rectangle` geerbten Datenfeldern `px py width` und `height` zusätzlich über das Datenfeld `my-color` . Dazu gibt es zwei Konstruktoren. Der erste initialisiert alle Datenfelder mit null, der zweite weist zusätzlich dem Datenfeld `my-color` eine Farbe zu.

Kapselung

Ein wesentlicher Vorzug objektorientierter Programmierung besteht darin, dass die internen Datenstrukturen einer Klasse verborgen sind. Zugriffe auf diese Strukturen sind im Allgemeinen nur über eine Schnittstelle möglich, die von den Methoden der Klasse bereitgestellt wird. Deshalb sind die Datenfelder einer Klasse normalerweise nur innerhalb der eigenen Klassendefinition und eventuell innerhalb der Klassendefinitionen der Nachfahren der Klasse sichtbar. In einem objektorientierten Forth kann man das erreichen, indem die Datenfelder und intern benutzte Methoden, die außerhalb der Klassendefinition nicht verwendet werden sollen, in besonderen Wortlisten gespeichert werden.

Diese Wortlisten heißen in Anlehnung an die von C++ benutzten Bezeichnungen `private` und `protected` . Wörter in der Wortliste `private` sind ausschließlich innerhalb der eigenen Klassendefinition sichtbar, während Wörter in der Wortliste `protected` zusätzlich von abgeleiteten Klassen benutzt werden können. Um die Datenfelder zu kapseln, könnte die Klasse `rectangle` folgendermaßen neu definiert werden:

```
class rectangle
  protected definitions
  null signed member px
  null signed member py
  null unsigned member width
  null unsigned member height
  forth definitions
  : rectangle ( rectangle -- 1st )
    dup erase ;
  : set-location ( signed signed rectangle -- )
    tuck py ! px ! ;
  : get-location ( rectangle -- signed signed )
    dup px @ swap py @ ;
  : set-size ( unsigned unsigned rectangle -- )
    tuck height ! width ! ;
  : get-size ( rectangle -- unsigned unsigned )
    dup width @ swap height @ ;
  : move ( signed signed rectangle -- )
    tuck py +! px +! ;
endclass
```

Sämtliche Datenfelder landen in der Wortliste `protected` , während die Methoden einschließlich des Konstruktors wieder Bestandteile der Wortliste `forth` sind. Den Zugriff auf die Datenfelder gewähren die vier neuen Methoden `set-location get-location set-size` und `get-size` . Würde später die Entscheidung getroffen, anstelle von Höhe und Breite die Koordinaten der oberen rechte Ecke des Rechtecks in den Datenfeldern zu speichern, müssten lediglich die Methoden `get-size` und `set-size` angepasst werden.

Die Wortliste `protected` wurde hier anstelle der Wortliste `private` benutzt, damit auf die Datenfelder auch in den Klassendefinitionen der von `rectangle` abgeleiteten Klassen, wie etwa `colored-rectangle` , zugegriffen werden kann. Die Wortliste `protected` wird als ein internes Attribut der Klasse `rectangle` gespeichert. Sie braucht nach außen nicht sichtbar zu sein. Damit sich verschiedene Klassen nicht in die Quere kommen, muss in jeder Klassendefinition eine neue Wortliste `protected` angelegt werden. Die Wortliste `protected` der Elternklasse wird übernommen und gegebenenfalls ergänzt.

Die bisher vorgestellten Konzepte zeigen nur die wichtigsten Grundlagen auf. Tatsächlich können auch fortgeschrittenere Techniken in einem System mit statischer Typprüfung implementiert werden. Hier ist eine Auswahl:

- Statische Speicherallokation für Objekte
- Datenfelder mit Character–Größe sowie Bitfelder
- Arrays als Datenfelder
- Vereinigungen (*unions*) von Datenfeldern
- Zugriff auf befreundete Klassen
- Kreuzreferenzen zwischen Klassen
- *early binding*

Die Klasse `input-stream`

Mit dem vorgestellten Handwerkszeug ist es bereits möglich, Objektorientierung in die Kernfunktionen eines Forth–Systems mit statischer Typprüfung einzuführen. Als ein Beispiel wird hier die Klasse `input-stream` vorgestellt.

Forth 2012 kennt als Quellen für den Interpreter die Konsole, Strings, Files und Blöcke. Die jeweilige Quelle wird anhand von `SOURCE-ID` und `BLK` unterschieden. Weitere Quellen kennt Forth 2012 nicht. Das Ganze ist historisch gewachsen und deshalb leider recht unsystematisch und wenig flexibel.

Ein alternativer, auf Objektorientierung basierender Ansatz definiert eine Klasse `input-stream`. Die Konsole, Strings, Files und Blöcke sind an Objekte dieser Klasse gebunden. Eines dieser Objekte ist in der Systemvariablen `default-input-stream` abgelegt, so dass der Interpreter zu jeder Zeit weiß, auf welche Quelle er zugreifen soll. Die Klasse `input-stream` verfügt über eine Anzahl teilweise virtueller Methoden, mit denen auf die Quelle zugegriffen werden kann. Jede Quelle, auch andere als die vier in Forth 2012 spezifizierten, kann somit eigene Varianten von `source >in refill save-input` und `restore-input` definieren, ohne dass es nötig ist, die gesamte Funktionalität von Anfang an in jeweils einer einzigen Fassung unterzubringen. Hier ist die Klassendefinition:

```
dt object procreates input-stream
class input-stream
  forth definitions
  virtual refill ( input-stream -- flag )
  virtual save-input ( input-stream -- 1st )
  virtual restore-input
  ( input-stream 1st -- flag )
  protected definitions
  null caddress -> character member 'buffer
  null unsigned member #buffer
  null unsigned member /buffer
  forth definitions
  null unsigned member >in
  : input-stream
    ( unsigned input-stream -- 2nd )
    locals| this | this erase
    dup this /buffer !
    chars callocate -> character this 'buffer !
    this ;
  : input-stream ( input-stream 1st -- 1st )
    locals| this | this copy
    0 this /buffer ! this ;
```

```
  : source
    ( input-stream --
      caddress -> character unsigned )
    locals| this |
    this 'buffer @
    this #buffer @ ;
  :noname ( input-stream -- flag )
    drop false ; is refill
  :noname ( input-stream -- 1st )
    new input-stream ; is save-input
  :noname ( input-stream 1st -- flag )
    locals| this |
    dup 'buffer @ this 'buffer @ =
    if dup >in @ this >in !
       dup #buffer @ this #buffer ! false
    else true
    then swap delete ; is restore-input
  :noname ( input-stream -- )
    locals| this | this /buffer @
    if this 'buffer @ free
    then [parent] delete ; is delete
endclass
```

Statt auf alle Details dieser Klassendefinition einzugehen, sollen hier nur die wichtigsten Aspekte erläutert werden. Die drei Datenfelder sind gekapselt, weil die Methoden alle erforderlichen Zugriffe darauf abdecken. `'buffer` enthält die Adresse des Lesepuffers, `#buffer` dessen Füllstand und `/buffer` die Puffergröße. Es gibt zwei Konstruktoren. Der erste legt einen neuen Lesepuffer mit einer gegebenen Größe an, während der zweite den Lesepuffer eines bereits existierenden Objekts der gleichen Klasse übernimmt und dabei `/buffer` auf null setzt als Hinweis, dass dieser Lesepuffer nur vom Originalobjekt gelöscht werden darf.

Jedes Objekt der Klasse `input-stream` hat einen eigenen Pufferzeiger `>in` . `source` ist eine normale Methode, während `refill save-input` und `restore-input` virtuelle Methoden sind. `save-input` erzeugt mit Hilfe des zweiten Konstruktors eine Kopie des Objekts, die von `restore-input` benutzt wird, um die ursprünglichen Datenfelder wiederherzustellen, sofern dies möglich ist. Am Ende von `restore-input` wird die Kopie gelöscht. Dazu wird die Methode `delete` der Elternklasse `object` verwendet, die von `[parent]` kompiliert wird. Ohne `[parent]` würde sich die virtuelle Methode der Klasse `input-stream` selbst aufrufen. Es entstünde eine endlose Rekursion.

Die virtuelle Methode `refill` liefert grundsätzlich ein `false` Flag, weil Objekten der Klasse `input-stream` keine konkrete Quelle zugeordnet ist. Sie ist eine Elternklasse. Dennoch verfügt sie über Konstruktoren, damit die von abgeleiteten Klassen wiederverwendet werden können. Eine solche abgeleitete Klasse ist `terminal-input-stream` :

```
dt input-stream procreates terminal-input-stream
class terminal-input-stream
  forth definitions protected
  : terminal-input-stream ( unsigned
    terminal-input-stream -- 2nd )
    input-stream ;
  : terminal-input-stream
    ( terminal-input-stream 1st -- 1st )
    input-stream ;
  :noname ( terminal-input-stream -- flag )
    locals| this |
    this 'buffer @ this /buffer @
    accept this #buffer !
    0 this >in ! true ; is refill
endclass
```

Diese Klasse benutzt die Konsole als Quelle. Die virtuelle Methode `refill` holt sich mittels `accept` eine neue Eingabezeile von der Konsole in ihren Lesepuffer.

`string-input-stream` ist ebenfalls von `input-stream` abgeleitet:

```
dt input-stream procreates string-input-stream
class string-input-stream
   forth definitions protected
   : string-input-stream
     ( caddress -> character unsigned
     string-input-stream -- 4 th )
     locals| this | this erase
     this #buffer ! this 'buffer ! this ;
   : string-input-stream
     ( string-input-stream 1st -- 1st )
     input-stream ;
endclass
```

Der zweite Konstruktor ist mit dem der Elternklasse identisch. Der erste Konstruktor erwartet zusätzlich zur Länge des Lesepuffers noch dessen Adresse auf dem Stack. Objekte der Klasse `string-input-stream` erstellen keinen eigenen Lesepuffer, sondern benutzen den Puffer, der dem Konstruktor angeboten wird. Damit die von der Elternklasse `input-stream` übernommene virtuelle Methode `delete` nicht versucht, den Speicherplatz des Puffers freizugeben, wenn das Objekt der Klasse `string-input-stream` gelöscht wird, bleibt das Datenfeld `/buffer` auf null. Alternativ hätte man für die Klasse `string-input-stream` eine eigene Version der virtuellen Methode `delete` definieren können. Da der Puffer nicht wieder aufgefüllt werden kann, übernimmt die Klasse die virtuelle Methode `refill` der Elternklasse `input-stream` , die das auch nicht kann.

Die Wörter `source >in refill save-input` und `restore-input` unterscheiden sich allerdings von den entsprechenden im Forth–2012–Standard spezifizierten Wörtern dadurch, dass sie jeweils ein Objekt der Klasse `input-stream` als zusätzlichen Eingangsparameter erfordern. Deshalb sollten überladene Versionen der fünf Wörter definiert werden, die sich auf die jeweils aktuelle Quelle beziehen und deshalb keinen zusätzlichen Eingangparameter benötigen:

```
user-input-device
cast input-stream variable default-input-stream
: >in ( -- address -> unsigned )
  default-input-stream @ >in ;
  1 retreat

: source ( -- caddress -> character unsigned )
  default-input-stream @ source ;
  1 retreat

: refill ( -- flag )
  default-input-stream @ refill ;
  1 retreat

: save-input ( -- input-stream )
  default-input-stream @ save-input ;
  1 retreat

: restore-input ( input-stream -- flag )
  default-input-stream @ restore-input ;
  1 retreat
```

Der Type–Cast vor der Variablendefinition ist erforderlich, weil die Konstante `user-input-device` den Datentyp `terminal-input-source` hat. Aber was hat es mit der Anweisung

```
  1 retreat
```

am Ende jeder Definition auf sich? Würde man beispielsweise `>in` ohne diese Anweisung definieren, könnte der Interpreter die erste Version von `>in` , die einen Parameter vom Datentyp `input-stream` erwartet, nicht mehr in der Wortliste finden. Der Interpreter würde als erstes die Version ohne Eingangsparameter finden und diese verwenden, selbst wenn sich ein Objekt vom Datentyp `input-source` auf dem Stack befindet. Deshalb ist es nötig, die Reihenfolge der beiden Versionen in der Wortliste zu vertauschen. `retreat` verbiegt die Verkettung der Wörter in der Wortliste. Es entfernt das zuletzt definierte Wort vom Ende der Wortliste und fügt es unmittelbar vor dem ersten Wort mit dem gleichen Namen ein, das es in der Wortliste findet. In diesem Fall handelt es sich um die Version von `>in` , die Bestandteil der Klassendefinition von `input-source` ist. Das neue Wort tritt gewissermaßen in der Wortliste um eine Position zurück. Der Parameter von `retreat` gibt an, um wie viele Positionen in Bezug auf gleichnamige Wörter das neue Wort zurückgestellt werden soll. Die Notwendigkeit der Existenz von `retreat` besteht nur in einem Forth–System mit statischer Typprüfung, die es erlaubt, Wörter zu überladen.

Des Weiteren müssen die Klassen `file-input-stream` und `block-input-stream` von `input-stream` abgeleitet werden. Die Klassendefinitionen werden hier nicht gezeigt, weil sie den Rahmen dieses Artikels sprengen würden. Stellt sich heraus, dass der Interpreter darüber hinaus noch andere Quellen erschließen muss, können problemlos weitere Klassen von `input-stream` abgeleitet werden. Forth 2012 bietet keine vergleichbaren Möglichkeiten. Die objektorientierte Herangehensweise und vor allem die Einbettung objektorientierter Techniken in ein System mit statischer Typprüfung erweist sich als weitaus flexibler.

Zusammenfassung

Die Integration objektorientierter Methoden in den Kern eines Forth–Systems erhöht sowohl die Flexibilität als auch die Übersichtlichkeit. Es ist deshalb naheliegend, ein neu zu entwickelndes System mit statischer Typprüfung von Beginn an objektorientiert zu entwerfen. Die Forderung nach Kompatibilität mit dem existierenden Standard kann ohnehin nicht erfüllt werden.

Die Klasse `input-stream` ist lediglich ein Beispiel. Es ist sinnvoll, weitere Datenstrukturen eines Forth–Systems mit statischer Typprüfung als Klassen zu implementieren. Dazu gehören neben einer Klasse `output-stream` auch Klassen für Wörter, Wortlisten, Stackdiagramme, Speicherbereiche, Kontrollfluss–Strukturen, Ausnahmebehandlungs–Strukturen und natürlich Datentypen.

Verweise

S. Becher, Statische Typprüfung in Forth, 4d2024–01.

`https://www.stephan-becher.de/strongforth3`

Die Zusammenarbeit von KI und Mensch am Beispiel eines Schachprogramms in Forth

Daniel Ciesinger, Cornu GmbH

Der Autor versucht, bei der Entwicklung eines Schachprogramms die Hilfe der künstlichen Intelligenz (KI) in Anspruch zu nehmen. Nutzen und Brauchbarkeit unterschiedlicher KIs für die Entwicklung von Forth–Quelltexten werden hier beleuchtet.

Motivation

Mit dem Aufkommen leistungsstarker Chat–KIs wie z. B. OpenAI Chat GPT, Mistral LeChat, Microsoft CoPilot, Deepseek R1, xAI Grok und Google Gemini wurde probiert, die Antworten der KI nicht nur in menschlicher Sprache, sondern auch in verschiedenen Computersprachen zu erzeugen.

Im Grundsatz würde eine programmierende KI zuerst einmal den Berufsstand des Programmierers wandeln, gleichzeitig wäre die KI aber auch in der Lage, reflektiv an sich selbst zu arbeiten.

Chat–KIs wurden bereits für einige Programmiersprachen zur Quelltexterzeugung genutzt und in Tools wie z. B. VSCode oder Jetbrains IDEA integriert.

Der Autor hat sich hier mit der Frage befasst, wie die KI die Erstellung von Forth–Quelltexten unterstützen kann und was dabei wissenswert ist.

Grundlagen

Derzeitige Chat–KIs basieren üblicherweise auf *erzeugenden, vorhersagenden Transformatoren*; sie versuchen also, auf der Basis einer Nutzereingabe eine Antwort zu erstellen, die zuallererst die Nutzereingabe enthält und dann davon ausgehend das jeweils nächste Wort der richtigen Antwort vorhersagt. Grundlage der Vorhersage ist eine Struktur neuronaler Netze mit einer Reihe unterschiedlicher Aufgaben. Die Trainingsdaten dieser Netze werden aus dem Internet und aus Nutzeraktionen gewonnen.

Die KI hat kein direkt greifbares und abrufbares Gedächtnis, sie kann aber lernen und in der Folge ähnliche Fragen unter Verwendung des Gelernten beantworten, was den Eindruck von Intelligenz erweckt.

Die gesamte Antwort wird sequentiell Wort für Wort (genauer: Silbe für Silbe) erzeugt, bei komplexeren Antworten kann der Anwender zusehen, wie die Antwort entsteht.

Die Programmiersprache Forth ist in ihrem Aufbau als durch Leerzeichen getrennte Wortfolge gut geeignet, von einer Chat–KI *verstanden* zu werden. Allerdings sind die meisten KIs mit umfangreichen Programmierbeispielen in ALGOL–ähnlichen Programmiersprachen trainiert und erwarten und erzeugen entsprechenden Code, oft von hoher cyclomatischer Komplexität, der sich in der Folge als nicht testbar erweisen kann. Bei einigen KIs kann man darauf hinweisen und um eine der Programmiersprache angemessene Implementierung bitten.

Die gezeigte Schachimplementierung ist als Versuch zu werten, gemeinsam mit der KI ein Programm in der Programmiersprache Forth zu erstellen.

Implementierung

Zum Zweck der Implementierung wurden der KI zahlreiche Aufgaben gestellt und Besonderheiten der Programmiersprache Forth *erklärt* sowie Lösungen der KI bewertet.

Schachbrett ausgeben

Im ersten Schritt wurde diese Anweisung eingegeben:

Schreibe in der Programmiersprache Forth ein Programm, das ein leeres Schachbrett als ASCII–Grafik auf der Konsole ausgibt.

xAI GROK Das Schachbrett wird auf Anhieb korrekt erzeugt, die Implementierung ist kurz und gut mit 2 ineinandergeschachtelten Schleifen.

Deepseek R1 Bis auf ein zweites # identisch mit GROK.

CoPilot Das Schachbrett wird auf Anhieb korrekt erzeugt, die Implementierung ist kurz und gut mit einer Schleife und den Zeilen als abwechselnde Zeichenketten.

Gemini 2.0 Flash erzeugt ein Schachbrett, das nur schwarze Felder zeigt. Das Programm ist gut strukturiert und im Forth–Stil mit deutschen Worten und Kommentaren geschrieben.

ChatGPT erzeugt ein Schachbrett, das nur weiße Felder zeigt. Nach einer Korrektur zeigt es schwarze und weiße Felder, aber nicht zeilenweise abwechselnd. Der Code ist Pascal–ähnlich strukturiert, englisch, ohne Kommentare.

LeChat erzeugt ein Schachbrett, das nur schwarze Felder zeigt, der Code ist Pascal–ähnlich strukturiert, englisch, ohne Kommentare.

GROK: Schachbrett ausgeben

```
: SCHACHBRETT ( - )
  8 0 DO
    8 0 DO
```

```
      I J + 2 MOD
      0= IF ." #" ELSE ." ." THEN
    LOOP CR
  LOOP ;
```

Schachbrett mit Figuren belegen

Im zweiten Schritt sollte das Schachbrett mit `CREATE` angelegt und als Zeichenkette repräsentiert werden:

Bonusaufgabe: Wir erstellen mit create ein virtuelles Schachbrett und setzen dort die Figuren in Form ihrer Anfangsbuchstaben, Schwarz in Kleinbuchstaben, Weiß in Großbuchstaben, leere Felder 0, A1 wäre Feld 0 mit Wert T (weißer Turm).

xAI GROK Das Schachbrett wird mit `CREATE` und `ALLOT` korrekt angelegt, jedes Feld wird mit c! belegt, allerdings fehlt jeweils das ASCII–Zeichen der Figur. Nach Korrekturhinweis werden die Figuren korrekt eingetragen. Der Verbesserungsvorschlag, `MOVE` zu verwenden, führt zu einigen Fehlern.

Deepseek R1 Das Schachbrett wird richtig erzeugt und in einer Init–Funktion mit den Figuren als String belegt.

CoPilot Das Schachbrett wird richtig erzeugt und direkt mit Strings belegt.

Gemini 2.0 Flash erzeugt mit `CREATE Schachbrett` einen Speicherbereich und belegt mit `ALLOT` 64 Byte darin. Danach wird eine Funktion definiert, die ein einzelnes Zeichen mit einem Offset platziert und mit dieser Funktion werden sämtliche Zeichen platziert. Es fehlt ein `SWAP`, in der Folge wird versucht, den Buchstaben als Adresse zu verwenden. Nach der Korrektur wird das Schachbrett bis auf die schwarzen Offiziere korrekt belegt. Der Verbesserungsvorschlag, `MOVE` mit String zu verwenden, wird gut ausgeführt.

ChatGPT erzeugt absolut unbrauchbaren Code, das `CREATE` fehlt, `ALLOT` wird in einer `:`–Definition ohne `CREATE` verwendet und es folgt ein großes `CASE`–Statement, das Buchstaben Zahlen zuordnet, aber nichts im Speicher ablegt. Der Korrekturwunsch (`CREATE` anstelle `:`, `MOVE` mit String) wird gut ausgeführt.

LeChat erzeugt teils sinnvollen Code. Wenn man auf die Ausführung der fehlerhaften Init–Funktion verzichtet, wird das korrekt mit `CREATE` und `ALLOT` angelegte Brett richtig mit Figuren belegt. Der Verbesserungswunsch wird gut ausgeführt.

CoPilot: Schachbrett belegen

```
create Belegung 64 allot
Belegung 64 0 fill
s" TSLDKLSTBBBBBBBB"
Belegung swap cmove
s" bbbbbbbbtsldklst"
Belegung 44 + swap cmove
```

Für die folgenden Aufgaben stand CoPilot nicht zur Verfügung, da der vorhandene Windows–PC anderweitig genutzt wurde.

Assoziativspeicher

Im dritten Schritt soll ein Wort zur assoziativen Speicherung geschrieben werden:

Schreibe ein Definitionswort in der Programmiersprache Forth, das bei Definition eine auf dem Stack übergebene Zeichenkette ablegt und bei Aufruf einen übergebenen Buchstaben in dieser Zeichenkette sucht und dessen Position zurückgibt. Anwendungsbeispiel: s" TSLDKBbdklst" assoziativ: Figurentabelle char T Figurentabelle .

xAI GROK erzeugt ein `CREATE-DOES>`–Wort mit `CREATE DUP , HERE SWAP CMOVE`, allerdings fehlt das `ALLOT`. Der `DOES>`–Teil sieht gut aus, führt aber zu einem Speicherfehler. Nach 25 Korrektureingaben endet die Sitzung ohne Erfolg.

Deepseek R1 erzeugt ein `CREATE-DOES>`–Wort. Der `CREATE`–Teil verursacht trotz weitgehend korrekter Verwendung von `HERE`, `ALLOT` und `CMOVE` mehrere Stackfehler. Der `DOES>`–Teil sieht plausibel aus, hat aber ebenfalls Stackfehler. Korrekturversuche führen zum Sitzungsabbruch wegen Überlastung.

Gemini 2.0 Flash erzeugt ein `CREATE-DOES>`–Wort mit `ALLOT` auf Basis der Stringlänge, es fehlt das Einkopieren (`PLACE`) des Strings. Der `DOES>`–Teil verwendet ein magisches Wort `BOUNDS` und ist auch ansonsten wenig sinnvoll, beabsichtigt aber das Richtige. Der Korrekturwunsch, `PLACE` im `CREATE`–Teil zu verwenden und `SCAN` im `DOES>`–Teil zu verwenden, wird recht gut ausgeführt. Es bleiben kleinere, aber sehr hartnäckige Fehler.

ChatGPT erzeugt ein `CREATE`–Wort mit `ALLOT`, allerdings wird versucht, die Adresse als Argument an `ALLOT` zu übergeben. Außerdem fehlt der `DOES>`–Teil, dieser wird als separate Funktion erzeugt. Entsprechende Korrekturwünsche führen nicht zum Erfolg.

LeChat erzeugt `CREATE-DOES>` ohne `ALLOT`, der `DOES>`–Teil arbeitet zeichenweise. Der Korrekturwunsch, `PLACE` im `CREATE`–Teil zu verwenden und `SCAN` im `DOES>`–Teil zu verwenden, wird recht gut ausgeführt, allerdings fehlt weiterhin ein `ALLOT`.

Gemini: Assoziativspeicher

```
: string-assoziativ ( addr u - )
  create
    here
    over 1+ allot
    place
  does> ( char - n )
    count dup >r rot scan
    r> swap dup if
      - nip
    else
      drop drop drop -1
    then ;
```

Zwischenfazit und weitere Schritte

Alle hier vorgestellten KIs wurden auf Deutsch angesprochen und haben dies einwandfrei verarbeitet. Ebenso konnten alle KIs compilierbare Quelltexte in Forth erzeugen. In vielen Fällen traten bei der Ausführung der Quelltexte Stackfehler auf. In einigen dieser Fälle half es, Stackdiagramme im Quelltext anzufordern und Debug–Ausgaben an die KIs zu übergeben.

CoPilot ist der *neugierige* KI–Assistent unter Windows. Er liefert gute Ergebnisse in Forth und ist auf aktuellen Windows–PCs gut nutzbar. Aus Sicherheitsgründen verwendet die Cornu GmbH keine eigenen Windows–PCs, diese werden üblicherweise von Kunden gestellt und gewartet.

Deepseek R1 hat als leistungsstarke *chinesische* KI für einigen Wirbel gesorgt, die Leistungen sind ordentlich. Ob der Medienrummel gerechtfertigt ist, darf zumindest aus Sicht von Forth bezweifelt werden.

ChatGPT und **LeChat** erwecken den Anschein, sie würden nur vorhandene Schachprogramme aus anderen Sprachen nach Forth übersetzen.

GROK macht nach 25 Eingaben eine Pause von 2 Stunden, es scheint also die erste *gewerkschaftlich organisierte KI* zu sein, Elon wird das nicht gefallen. GROK hat trotzdem sehr viel Potential und reagiert gut auf Korrekturwünsche.

Weitere Aufgabenstellungen an die Gemini AI waren

- Die Schwarz–Weiß–Darstellung des Schachfeldes über ANSI–Escape–Sequenzen — gute Unterstützung.
- Die Verwendung der UTF–8–Schachfigurensymbole — sehr gute Unterstützung.
- Die Erstellung eines Recognizers — das Konzept *versteht* die KI nicht, selbst nach mehrfacher Erklärung.
- Objektorientierte oder polymorphe Schachfiguren — hier ist die KI nützlich und gibt gute Hinweise.
- Erstellung eines Computergegners — hier liefert die KI eine Pascal–ähnliche Min–Max–Lösung ab, verwendet allerdings bestehenden Code. Nach Aufforderung wird hier ein Refactoring vorgenommen und der Code wird deutlich besser.

Gemini ist eine sehr wortreiche, teils langatmige KI, sowohl im Umfang ihrer Antworten als auch bezogen auf die erzeugten Quelltexte. Im Gegensatz dazu ist z. B. GROK recht wortkarg.

Es ist anzumerken, dass die Interaktion mit der KI das Programmieren auch hinsichtlich der Arbeitsmoral unterstützt, denn durch den Dialog bleibt man leichter in die Aufgabe involviert und denkt mehr über Lösungen nach, insbesondere solche, die eher Forth–typisch sind als die von der KI vorgeschlagenen.

Die Gemini–KI ist ein überaus angenehmer Gesprächspartner mit guter deutscher Wortwahl und ausgezeichneter Höflichkeit und Geduld. Als wesentliches Manko bleibt das offensichtlich unbeabsichtigte Beharren auf falschen Lösungen insbesondere in den sogenannten Code–Snippets. Es scheint, als würde die KI den falschen Code nicht *sehen* oder entdecken. Eine weitere Schwäche ist, dass Korrekturen gelegentlich additiv erfolgen, zum bestehenden fehlerhaften Code wird weiterer Code hinzugefügt, was teilweise den Code derart verschlechtert, dass die KI zum Neustart der Aufgabe aufgefordert werden muss. So kann z. B. der Versuch, ein falsches `SWAP` anstelle eines `ROT` zu korrigieren, dazu führen, dass ein weiteres `SWAP` eingefügt wird. Den Hinweis an die KI, dass `SWAP SWAP` einem NOP entspricht, bestätigt die KI, verwendet aber weiterhin `SWAP SWAP`, anstatt dieses wegzulassen.

Interessant ist auch *magisches Denken*, so fabuliert die KI Hilfsworte herbei, die passend aussehen oder sie erklärt ausführlich, warum `-1 +LOOP` nicht das Gleiche ist wie die Abfolge von `-1` und `+LOOP`. In den FAQ zu Gemini wird auch von *Halluzinationen* gesprochen, die die KI dazu bewegen, unerwartet die intellektuell ausgetreten Pfade der Vernunft zu verlassen und auf der wilden Wiese des Wahnsinns herumzutollen.

Gemini Flash 2.0 erscheint knapp so *intelligent*, dass eine Zusammenarbeit eher Vorteile bringt. Andere KIs können eventuell eine geringere Leistungsfähigkeit haben, von einem schlechteren Anfangsniveau starten oder Korrekturen weniger gut umsetzen oder diese zu leicht *vergessen*. Dies kann sehr ernüchternd sein und dem Anwender die Freude an der Zusammenarbeit nehmen.

Ausblick

Um die KI mit Forth wirklich gut nutzen zu können, wären folgende Dinge wichtig:

- Eine von der KI nutzbare Debug–Umgebung, damit die KI den von ihr erstellten Code prüfen kann.
- Ein an die KI gekoppelter Editor. Damit kann z. B. vom Anwender ein Kommentar geschrieben werden und die KI erzeugt den passenden Code.
- Erstellen einer KI in Forth.

Als Verbesserungen des Schachprogramms stehen folgende Themen an:

- Rochade (o–o, o–o–o, O–O, O–O–O) unter Verwendung der jeweiligen Bedrohungskarte, um festzustellen, ob eines der Felder angegriffen ist. Außerdem ist hier ein Gedächtnis erforderlich, da sich der König vorher nicht bewegt haben darf.
- En Passant — hier ist ebenfalls ein Gedächtnis erforderlich, um den vorherigen Zug des gegnerischen Bauern zu prüfen.
- Umwandlung von Bauern an der gegnerischen Grundlinie
- Schacherkennung und Matt–Erkennung
- Computerspieler, hierbei könnten die verschiedenen Karten hilfreich sein.

Referenzen

1. Google Gemini `https://gemini.google.com/`
2. xAI GROK `https://x.com/i/grok`
3. Deepseek R1 `https://www.deepseek.com/a/chat`
4. Ein Schachprogramm in Forth `https://www.quirkster.com/iano/forth/FCP.html`

Listing

```
\ Schachspiel
\ Urheber: Daniel Ciesinger, Cornu GmbH 2025 mit Hilfe verschiedener Chat-KIs http://www.cornu.de

8 constant breite
8 constant zeilen

\ Das Schachbrett
create Belegung 65 allot
Belegung 65 bl fill
s" TSLDKLSTBBBBBBBB" Belegung 1+   swap move
s" bbbbbbbbtsldklst" belegung 49 + swap move

\ ======== Karten zur Situationsanalyse ============

\ Die Bedrohungskarte zeigt, ob Bedrohungen ausgeglichen sind
\ Die Werte sind -8...+8 mit 1/8 skaliert, die Null hat den Wert 64
create Bedrohungskarte   65 allot
Bedrohungskarte 65 64 fill

\ Von Weiß bedrohte Felder
create Bedrohungskarte_w   65 allot
Bedrohungskarte_w 65 0 fill

\ Von Schwarz bedrohte Felder
create Bedrohungskarte_s   65 allot
Bedrohungskarte_s 65 0 fill

\ Die Gefechtskarte zeigt, welche Felder besonders stark angegriffen UND verteidigt sind
\ Die Werte sind 0...+16 mit 1/8 skaliert, die Null hat den Wert 0
create Gefechtskarte 65 allot
Gefechtskarte 65 0 fill

\ ======== Assoziativspeicher ===========
: assoziativ:  ( addr-c len --> name)
    CREATE
    HERE
    over dup >R
    1+ allot
    place
    R> 1+ cells allot
  DOES> ( char -- addr )
    count
    2dup + >R
    dup >R rot scan R> swap
    dup IF
        - 1+ swap drop cells R> +
    ELSE 2drop drop R>
    THEN
;

\ Jede Schachfigur anlegen
s" TSLDKBbdklst" assoziativ: Figurentabelle

\ ========== Schachbrett ausgeben mit ANSI Escape-Sequenzen =============
: .esc  27 emit ;                                   \ Escape-Zeichen ausgeben
```

```
: schwarzes-feld  .esc ." [40m" ;    \ ANSI schwarzer Hintergrund
: weisses-feld    .esc ." [47m" ;    \ ANSI weißer    Hintergrund
: schwarze-figur  .esc ." [30m" ;    \ ANSI schwarze  Schrift
: graue-figur     .esc ." [30;1m" ;  \ ANSI graue     Schrift
: weisse-figur    .esc ." [37;1m" ;  \ ANSI weiße     Schrift
: roter-punkt     .esc ." [31;1m." ;
: gruener-punkt   .esc ." [32;1m." ;
: zuruecksetzen   .esc ." [0m" ;     \ ANSI Zurücksetzen
: beschriftung ."    A  B  C  D  E  F  G  H  " cr ; \ Spaltenbeschriftung

\ Weiße Figur ausgebe
: <weiss ( koord feldfarbe -- koord )
    IF weisses-feld ELSE schwarzes-feld THEN ( koord )
    dup bedrohungskarte_s + c@ 0> IF roter-punkt ELSE space THEN
    weisse-figur ;
: weiss> ( koord -- )
    bedrohungskarte_w + c@ 0> IF gruener-punkt ELSE space THEN zuruecksetzen ;

\ Schwarze Figur ausgeben
: <schwarz ( koord feldfarbe -- koord )
    IF
        weisses-feld
        dup bedrohungskarte_w + c@ 0> IF roter-punkt ELSE space THEN
        schwarze-figur
    ELSE
        schwarzes-feld
        dup bedrohungskarte_w + c@ 0> IF roter-punkt ELSE space THEN
        graue-figur
    THEN ( koord )
;
: schwarz> ( koord -- )
    bedrohungskarte_s + c@ 0> IF gruener-punkt ELSE space THEN zuruecksetzen ;

\ Figuren ausgeben als UTF-8-Schachfigur
HEX
: .Leer ( koord feldfarbe_w -- )
  IF weisses-feld ELSE schwarzes-feld THEN 3 spaces drop zuruecksetzen ;
: .KW   <weiss   265A XEMIT weiss> ;
: .KS   <schwarz 265A XEMIT schwarz> ;
: .DW   <weiss   265B XEMIT weiss> ;
: .DS   <schwarz 265B XEMIT schwarz> ;
: .TW   <weiss   265C XEMIT weiss> ;
: .TS   <schwarz 265C XEMIT schwarz> ;
: .LW   <weiss   265D XEMIT weiss> ;
: .LS   <schwarz 265D XEMIT schwarz> ;
: .SW   <weiss   265E XEMIT weiss> ;
: .SS   <schwarz 265E XEMIT schwarz> ;
: .BW   <weiss   265F XEMIT weiss> ;
: .BS   <schwarz 265F XEMIT schwarz> ;
DECIMAL

\ ========== Gültigkeit von Zügen prüfen ===========
\ Ohne Berücksichtigung belegter Felder
\ Makros
: koord>xy  ( koord -- x y ) 1- breite /mod 1+ swap 1+ swap ;
: xy>koord  ( x y -- koord ) 1- breite * + ;
: delta ( koord koord -- dx dy ) swap koord>xy rot koord>xy rot - >R swap - R>  ;
: delta_abs ( koord koord -- dx dy ) delta abs swap abs swap  ;
: B_max ( delta_y koord grundlinie -- flag ) swap koord>xy nip = IF 1 3 within ELSE  1 = THEN ;

\ Zugprüfung für jede Figur
: ?XG  ( koord koord -- flag ) 2drop FALSE ; \ Leeres Feld bewegen ist immer ungültig
: ?KG  ( koord koord .. flag ) delta_abs max 1 = ;  \ Rochade TODO
: ?TG  ( koord koord -- flag ) delta_abs 2dup 0= swap 0> AND >R 0> swap 0= AND R> OR ;
: ?LG  ( koord koord -- flag ) delta_abs = ;
: ?SG  ( koord koord -- flag ) delta_abs 2dup 2 = swap 1 = AND >R 1 = swap 2 = AND R> OR ;
```

```
: ?BGW ( koord koord -- flag ) over >R delta swap 0= swap        R> 2 B_max  AND ; \ En Passant TODO
: ?BGS ( koord koord -- flag ) over >R delta swap 0= swap negate R> 7 B_max  AND ;
: ?BXW ( koord koord -- flag ) delta swap abs over = swap 1 = AND  ;
: ?BXS ( koord koord -- flag ) delta swap abs over negate = swap -1 = AND ;
: ?DG  ( koord koord .. flag ) 2dup ?TG >R ?LG R> OR ;

\ ========== Blockadefreiheit von Zügen prüfen ===========
\ Makros
: KeineFigur?     ( koord -- flag )   Belegung + c@ BL = ;
: Belegt?         ( koord -- flag )   KeineFigur? invert ;
: WeisseFigur?    ( koord -- flag )   Belegung + c@ c" TSLDKB" count rot scan nip 0<> ;
: SchwarzeFigur?  ( koord -- flag )   Belegung + c@ c" btsldk" count rot scan nip 0<> ;
: Zaehlschritt ( koord koord' -- schrittx schritty )
  delta ( dx dy )
  dup IF dup abs / ELSE drop 0 THEN \ dy anpassen
  swap dup IF dup abs / ELSE drop 0 THEN \ dx anpassen
  swap
;

: Pfadlaenge      ( koord koord' -- delta ) delta_abs max ;
: LeererPfad?     ( koord koord' schritte -- flag ) \ TRUE, wenn der Pfad leer ist.
    >R
    over swap Zaehlschritt   ( koord schrittx schritty R: schritte )
    R> 1 ?DO                 ( koord schrittx schritty )
        over I *             ( koord schrittx schritty schrittx*I )
        over I *             ( koord schrittx schritty schritty*I schritty*I )
        4 pick               ( koord schrittx schritty schrittx*I schritty*I koord )
        koord>xy             ( koord schrittx schritty schrittx*I schritty*I x y )
        rot + >R             ( koord schrittx schritty schrittx*I x R: y+schritty*I )
        swap + R>            ( koord schrittx schritty x+schrittx*I y+schritty*I )
        xy>koord Belegt?     ( koord schrittx schritty flag )
        IF rot drop FALSE -rot LEAVE THEN
    LOOP
    2drop 0<>
;

\ Prüfung auf "nicht blockiert" für jede Figur
: ?XB   ( koord koord -- flag ) 2drop FALSE ; \ Leeres Feld sind immer blockiert
\ Figuren, die einen Schritt gehen, auf "nicht blockiert" prüfen
: ?1BZ  ( koord koord .. flag ) nip KeineFigur? ;     \ Zug ohne Schlagen
: ?1BSW ( koord koord -- flag ) nip SchwarzeFigur? ;  \ Schlagzug der weißen Figur
: ?1BSS ( koord koord -- flag ) nip WeisseFigur? ;    \ Schlagzug der schwarzen Figur
\ Figuren, die "n" Schritt gehen, auf "nicht blockiert" prüfen
: ?NBZ  ( koord koord -- flag ) 2dup Pfadlaenge LeererPfad? ;
: ?NBSW ( koord koord -- flag ) dup >R 2dup Pfadlaenge 1- LeererPfad? R> SchwarzeFigur? AND ;
: ?NBSS ( koord koord -- flag ) dup >R 2dup Pfadlaenge 1- LeererPfad? R> WeisseFigur?   AND ;

\ ===== Berechne Bedrohung und Gefechtsintensität ======
: ImFeld? ( x y -- flag )   1 9 within >R   1 9 within R> AND ;
: Diagonal?   ( dx dy -- flag )   delta_abs = ;  \ TRUE für diagonal

\ Dämpfung berechnen - jede Figur im Weg dämpft die Kraftlinie, sofern sie diese nicht stärkt
: Daempfung ( dx dy koord farbe -- Wert' )
    dup 0= IF 2drop 2drop 0 EXIT THEN
    >R \ Farbe auf TOR
    Belegung + c@ -rot
    diagonal? IF \ diagonaler Pfad?
        dup [char] l R@ 0> IF toupper THEN = IF drop R> EXIT THEN
        dup [char] d R@ 0> IF toupper THEN = IF drop R> EXIT THEN
    ELSE
        dup [char] t R@ 0> IF toupper THEN = IF drop R> EXIT THEN
        dup [char] d R@ 0> IF toupper THEN = IF drop R> EXIT THEN
    THEN
    BL = IF R> EXIT THEN
    R@ abs 1 <= IF  R> drop 0 ELSE R> 2/ THEN
;
```

```
188
189  : Schritt ( dx dy koord -- x y )
190      koord>xy ( dx dy x y )
191      rot +  >R ( dx x R:  y+dy )
192      swap + R>
193  ;
194
195  : AktualisiereKarten ( koord farbe -- )
196      >R
197      dup Bedrohungskarte   + R@                  swap c+!
198      dup Gefechtskarte     + R@ abs              swap c+!
199      dup Bedrohungskarte_w + R@        8 >= IF 1 swap c!  ELSE drop THEN
200          Bedrohungskarte_s + R> negate 8 >= IF 1 swap c!  ELSE drop THEN
201  ;
202
203  \ Entlang des Pfades Werte addieren und Dämpfung berücksichtigen
204  : Pfad    ( koord dx dy farbe -- )
205      >R
206      BEGIN  ( koord dx dy  R: farbe )
207          rot >R 2dup R>   ( dx dy dx dy koord R: farbe )
208          Schritt          ( dx dy x y R: farbe )
209          2dup ImFeld?     ( dx dy x y R: farbe )
210          >R xy>koord R>   ( dx dy koord flag R: farbe )
211      WHILE                ( dx dy koord R: farbe )
212              dup R@ AktualisiereKarten ( dx dy koord R: farbe )
213              -rot 2dup 4 pick R> ( koord dx dy dx dy koord farbe )
214              Daempfung >R  ( koord dx dy R: farbe' )
215      REPEAT ( koord dx dy R: farbe )
216      R> 2drop 2drop
217  ;
218
219  : Nachbar ( koord dx dy farbe -- ) \ Addiere 1/-1 zu Nachbarfeld, keine Dämpfung
220      >R
221      rot Schritt
222      2dup ImFeld? IF ( x y  R: farbe )
223          xy>koord R> AktualisiereKarten
224      ELSE R> drop 2drop
225      THEN
226  ;
227
228  : XX drop ;
229  : LXW ( koord -- ) dup 1 1  8 Pfad       dup 1 -1  8 Pfad  dup  -1 1  8 Pfad    -1 -1  8 Pfad ;
230  : LXS ( koord -- ) dup 1 1 -8 Pfad       dup 1 -1 -8 Pfad  dup  -1 1 -8 Pfad    -1 -1 -8 Pfad ;
231  : TXW ( koord -- ) dup 1 0  8 Pfad       dup 0  1  8 Pfad  dup  -1 0  8 Pfad     0 -1  8 Pfad ;
232  : TXS ( koord -- ) dup 1 0 -8 Pfad       dup 0  1 -8 Pfad  dup  -1 0 -8 Pfad     0 -1 -8 Pfad ;
233  : DXW ( koord -- ) dup LXW TXW ;
234  : DXS ( koord -- ) dup LXS TXS ;
235  : KXW ( koord -- )
236      dup 1 1 8 Nachbar     dup -1 1 8   Nachbar  dup 1 -1 8 Nachbar   dup -1 -1 8  Nachbar
237      dup 1 0 8 Nachbar     dup -1 0 8   Nachbar  dup 0  1 8 Nachbar        0 -1 8  Nachbar ;
238  : KXS ( koord -- )
239      dup 1 1 -8 Nachbar    dup -1 1 -8  Nachbar  dup 1 -1 -8 Nachbar  dup -1 -1 -8 Nachbar
240      dup 1 0 -8 Nachbar    dup -1 0 -8  Nachbar  dup 0  1 -8 Nachbar       0 -1 -8 Nachbar ;
241  : SXW ( koord -- )
242      dup 2 1  8 Nachbar    dup -2  1  8 Nachbar  dup 2 -1  8 Nachbar  dup -2 -1  8 Nachbar
243      dup 1 2  8 Nachbar    dup -1  2  8 Nachbar  dup 1 -2  8 Nachbar      -1 -2  8 Nachbar ;
244  : SXS ( koord -- )
245      dup 2 1 -8 Nachbar    dup -2  1 -8 Nachbar  dup 2 -1 -8 Nachbar  dup -2 -1 -8 Nachbar
246      dup 1 2 -8 Nachbar    dup -1  2 -8 Nachbar  dup 1 -2 -8 Nachbar      -1 -2 -8 Nachbar ;
247  : BXW ( koord -- )
248      dup 1  1  8 Nachbar       -1  1  8 Nachbar ; \ Für Bedrohung sind nur Schlagzüge relevant
249  : BXS ( koord -- ) dup 1 -1 -8 Nachbar       -1 -1 -8 Nachbar ;
250
251
252
253
```

```
\ ======= Assoziativspeicher befüllen ================
\ Makros
: >FT                  ( n char -- )                 Figurentabelle ! ;
: ->Figurenwert@       ( figur -- n )                Figurentabelle @       @ ;
: ->Figurenfarbe@      ( figur -- farbe )            Figurentabelle @ cell+ @ ;
: ->Ausgeben           ( feldfarbe figur -- )        Figurentabelle @ 2 cells + @ execute ;
: ->Gueltig?           ( koord koord' figur -- flag ) Figurentabelle @ 3 cells + @ execute ;
: ->Gueltig_Schlagzug? ( koord koord' figur -- flag ) Figurentabelle @ 4 cells + @ execute ;
: ->Blockiert?         ( koord koord' figur -- flag ) Figurentabelle @ 5 cells + @ execute invert ;
: ->Blockiert_schlag?  ( koord koord' figur -- flag ) Figurentabelle @ 6 cells + @ execute invert ;
: ->Bedrohung          ( koord figur -- )            Figurentabelle @ 7 cells + @ execute ;

\ Eintragungen in der Figurentabelle
\      Figur   | Wert |Farbe | Ausgeben | gültig?        | blockiert?      | Bedrohung
\              |      |      |          | gehen schlagen | gehen schlagen  |
create Leer       0 ,  0 ,  ' .Leer , ' ?XG ,  ' ?XG ,  ' ?XB ,  ' ?XB ,   ' XX ,  Leer       char X >FT
create Koenig_w 100 ,  1 ,  ' .KW ,   ' ?KG ,  ' ?KG ,  ' ?1BZ , ' ?1BSW , ' KXW , Koenig_w   char K >FT
create Koenig_s 100 , -1 ,  ' .KS ,   ' ?KG ,  ' ?KG ,  ' ?1BZ , ' ?1BSS , ' KXS , Koenig_s   char k >FT
create Dame_w    10 ,  1 ,  ' .DW ,   ' ?DG ,  ' ?DG ,  ' ?NBZ , ' ?NBSW , ' DXW , Dame_w     char D >FT
create Dame_s    10 , -1 ,  ' .DS ,   ' ?DG ,  ' ?DG ,  ' ?NBZ , ' ?NBSS , ' DXS , Dame_s     char d >FT
create Turm_w     5 ,  1 ,  ' .TW ,   ' ?TG ,  ' ?TG ,  ' ?NBZ , ' ?NBSW , ' TXW , Turm_w     char T >FT
create Turm_s     5 , -1 ,  ' .TS ,   ' ?TG ,  ' ?TG ,  ' ?NBZ , ' ?NBSS , ' TXS , Turm_s     char t >FT
create Laeufer_w  3 ,  1 ,  ' .LW ,   ' ?LG ,  ' ?LG ,  ' ?NBZ , ' ?NBSW , ' LXW , Laeufer_w  char L >FT
create Laeufer_s  3 , -1 ,  ' .LS ,   ' ?LG ,  ' ?LG ,  ' ?NBZ , ' ?NBSS , ' LXS , Laeufer_s  char l >FT
create Springer_w 2 ,  1 ,  ' .SW ,   ' ?SG ,  ' ?SG ,  ' ?1BZ , ' ?1BSW , ' SXW , Springer_w char S >FT
create Springer_s 2 , -1 ,  ' .SS ,   ' ?SG ,  ' ?SG ,  ' ?1BZ , ' ?1BSS , ' SXS , Springer_s char s >FT
create Bauer_w    1 ,  1 ,  ' .BW ,   ' ?BGW , ' ?BXW , ' ?NBZ , ' ?1BSW , ' BXW , Bauer_w    char B >FT
create Bauer_s    1 , -1 ,  ' .BS ,   ' ?BGS , ' ?BXS , ' ?NBZ , ' ?1BSS , ' BXS , Bauer_s    char b >FT

\ ====== Schachbrett ausgeben, Teil 2 ==============
\ Einzelnes Schachfeld mit Figur und Farbe ausgeben

create bedr>farbe   46 c, 46 c, 44 c, 42 c, 47 c, 43 c, 41 c, 45 c, 45 c,
: .bedrohung ( n -- )
    dup 16 / bedr>farbe + c@ s>d .esc [char] [ emit <# # # #>  type ." ;37;1m"
    dup 64 >= IF 64 - space FALSE
    ELSE 64 swap - TRUE THEN
    swap  100 8 */ s>d
    <# # # [char] , hold # rot sign #>  type
    zuruecksetzen
;
create gef>farbe    47 c, 44 c, 46 c, 46 c,  42 c, 43 c, 41 c, 45 c, 45 c,
: .gefecht ( n -- )
    dup 16 / gef>farbe + c@ s>d .esc [char] [ emit <# # # #>  type ." ;37;1m"
    100 8 */
    dup 1000 < IF SPACE THEN
    s>d
    <# # # [char] , hold #s #>  type
    zuruecksetzen
;

: BedrohungEintragen ( -- )
    Bedrohungskarte 65 64 fill
    Gefechtskarte   65 0 fill
    Bedrohungskarte_w 65 0 fill
    Bedrohungskarte_s 65 0 fill
    66 1 DO I Belegung I + c@ ->Bedrohung LOOP
;

\ Schachbrett ausgeben
: .schachbrett ( -- )
    BedrohungEintragen
    1 Zeilen DO ( ) \ Schachbrett ausgeben
        cr I .
        Breite 1+ 1 ?DO
            I J xy>koord       ( koord )
```

```
320             I J + 2 MOD        ( koord farbe )
321             over Belegung + c@ ( koord farbe figur )
322             ->Ausgeben         ( )
323         LOOP
324         3 spaces
325         breite 1+ 1 ?DO ( ) \ Bedrohungskarte ausgeben
326             I J xy>koord Bedrohungskarte + c@ .bedrohung
327         LOOP
328         3 spaces
329         breite 1+ 1 ?DO ( ) \ Gefechtskarte ausgeben
330             I J xy>koord Gefechtskarte + c@ .gefecht
331         LOOP
332     -1 +LOOP
333     cr beschriftung
334 ;
335
336 \ === Züge in algebraischer Notation ======================
337
338 \ Rechne die Schachkoordinate in algebraischer Notation um in linearen Index
339 \ Gib 0 zurück, falls die Koordinate ungültig ist.
340 : koordinate ( c-addr u -- index|0 )
341     2 <> IF drop 0  EXIT THEN \ Länge prüfen, bei !=2 0 zurückgeben
342     dup        c@ dup [char] a [char] i WITHIN
343       IF  [char] a - 1+     ELSE  2drop 0 EXIT THEN \ Spalte
344     swap char+ c@ dup [char] 1 [char] 9 WITHIN
345       IF  [char] 1 - 8 * +  ELSE  2drop 0 EXIT THEN \ Zeile, Index
346 ;
347
348 \ Steht die in algebraischer Notation angegebene Figur tatsächlich auf dem Feld?
349 \ Format Fsz-sz mit F=Figur, sz=Koordinate und -= Trennzeichen vorher/nachher
350 : Figur?   ( c-addr u -- flag )
351     over c@ -rot
352     1 /string drop 2 koordinate Belegung + c@ =
353 ;
354
355 \ Das Trennzeichen - oder x entscheidet, ob gezogen oder geschlagen werden soll.
356 : Trennzeichen? ( char -- flag )
357     dup  [char] - =
358     swap [char] x = OR
359 ;
360
361 \ Schachzug in algebraischer Notation - prüfe das Format und berechne die Koordinaten
362 : zug? ( c-addr u -- koord koord' figur schlag? | 0 )
363     dup 6 = IF ( c-addr u )
364         2dup Figur? 0= IF 2drop FALSE EXIT THEN \ Figur steht nicht auf angegebenenm Feld?
365         over dup 3 + c@
366         dup Trennzeichen? 0= IF 2drop R> 2drop FALSE EXIT THEN \ falsches Trennzeichen?
367         >R c@ >R
368         1 /string
369         over 2    koordinate -rot \ Startkoordinate prüfen (mit u=2)
370         3 /string koordinate      \ Zielkoordinate prüfen (mit u=2)
371         over 0= over 0= OR IF 2drop R> R> 2drop FALSE EXIT THEN \ Fehler in Koordinaten?
372         R> R>
373     ELSE 2drop FALSE \ falsche Länge?
374     THEN
375 ;
376
377 \ Zug ausführen
378 : ziehen ( koord koord' figur -- )
379     >R
380     2dup R@ ->Gueltig?    0= IF 2drop R> drop  ." ungültiger Zug"     exit THEN
381     2dup R> ->Blockiert?     IF 2drop          ." Figur steht im Weg" exit THEN
382     swap Belegung + dup \ Startfeldadresse berechnen
383     c@ >R
384     BL swap c!   Belegung +    R> swap c!  ;
385
```

```
\ Zug ausführen und schlagen
: schlagen ( koord koord' figur -- )
    >R
    2dup R@ ->Gueltig_Schlagzug? 0= IF 2drop R> drop ." ungültiger Zug"     exit THEN
    2dup R> ->Blockiert_Schlag?     IF 2drop         ." Figur steht im Weg" exit THEN
    swap Belegung + dup \ Startfeldadresse berechnen
    c@ >R
    BL swap c!   Belegung +    R> swap c!  ;

\ ========== Recognizer für algebraische Notation =============

\ ===== Das fehlt in gforth 0.7 ==================
\ Recognizer zu Recognizer-Tabelle hinzufügen
: +RECOGNIZER ( xt -- )
    >R GET-RECOGNIZERS 1+ R> SWAP SET-RECOGNIZERS ;
\ rectype none
: RECTYPE-NONE rectype-null ;
\ ================================================

\ Recognizer-Typ für algebraische Notation
' EXECUTE       \ Interpret-Verhalten
' DROP          \ Compile-Verhalten
' DROP          \ Postpone-Verhalten
RECTYPE: RECTYPE-SCHACHZUG

\ Erkenne die algebraische Notation und führe den Zug aus
: REC-SCHACHZUG ( c-addr len -- i*x addr1 | addr2 )
    Zug? dup IF
        [char] - = IF ['] ziehen   RECTYPE-SCHACHZUG
        ELSE          ['] schlagen RECTYPE-SCHACHZUG
        THEN
    ELSE drop RECTYPE-NONE
    THEN ;

\ add the recognizer to the table of recognizers
' REC-SCHACHZUG +RECOGNIZER

TRUE constant Testmodus
Testmodus [IF]
.schachbrett  Bd2-d4 be7-e5 Bd4xe5  .schachbrett
[THEN]
```

Nach so viel Code muss noch ein Bild daher, um das Auge zu entspannen.

Da sich nichts Passendes zum Thema „Schachspielen in Forth“ fand und hier sowieso KI–Hilfe in Anspruch genommen wurde, hab ich Gemini gebeten, ein „picture of doubleheads dragon playing chess“ zu malen. Hat nur einige Sekunden gedauert.

Wie es aussieht, sind doppelköpfige Drachen in der KI alles andere als niedlich. Doch hier faltet er brav die Hände und wartet, bis er wieder an Zug ist, da in dem antiken Gewölbe.

Swappy erscheint da sehr mächtig. Was Forth ja auch ist. Und es ist nicht so leicht gewesen, das Monster zu zähmen und auf Schach abzurichten. Doch Daniel ist es gelungen. Den Code konnte ich noch nicht ausprobieren — ihr? Schreibt mir doch über eure Erfahrung.

Übrigens: Das PDF vom Heft ist durchgängig farbig! Blick ins Archiv der VD im Forth–ev–Wiki lohnt sich. Ich finde das Bild super! :) mk

Breadboards III

Rafael Deliano

Messtechnik an Versuchsaufbauten soll trotz widriger Umstände sinnvolle Ergebnisse lieferen. Dafür ist vieles zu beachten.

EMV[1]

Es werden meist mehrere Geräte extern angeschlossen, die Folge sind Störsignale. Das ist ungünstig für analoge Schaltungen wie Sensoren mit kleinem Signalpegel.

Ursache der Störsignale sind häufig Masse–Probleme (Abb. 1). Früher waren da nur Brummschleifen, d. h. 50 Hz und Oberwellen üblich, das Problem damit klar identifizierbar. Heute ist es ein breitbandiger Rauschteppich meist kleiner Spikes, gegen den man nur mit einer Vielzahl von Maßnahmen ankommt.

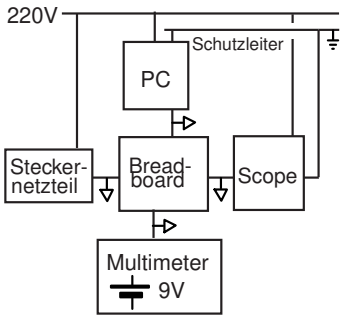


Abbildung 1: Typischer Aufbau

Erstens sollten alle Geräte über eine gemeinsame Steckdosenleiste versorgt werden. Günstig sind batteriebetriebene Messgeräte wie Multimeter, sie ermöglichen floatende Messung überall in der Schaltung.

Kleine DC–Steckernetzteile, z. B. 12 V, ohne Schutzleiter, sind nominell galvanisch getrennt und hätten damit Eigenschaften wie eine Batterie. Tatsächlich sind es aber heute Schaltregler, deren Trafo durch einen kleinen Folienkondensator überbrückt wird, damit sie ihren EMV–Test bestehen (Abb. 2). Man koppelt also hohe Frequenzen aus dem Netz ein. Alternative kann ein altes 50Hz–Steckernetzteil mit 2–Kammer–Wicklung und Längsregler sein. In extremen Fällen verwendet man kleine 12V–Bleibatterien, wie sie für Schiffsmodelle üblich waren.[2]

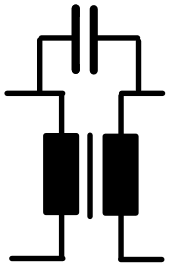


Abbildung 2: Trafo in Steckernetzteil mit DCDC–Wandler

Bei Geräten wie PCs oder Oszilloskopen ist das Metallgehäuse auf Schutzleiter gelegt. Dort liegt dann auch der GND des Sub–D–Steckers, des USB, der BNC–Buchse. Und damit der GND des Breadboards.

Akkubetriebene Laptops helfen da nur bedingt. Aber weil die Verbindung zum Controller oft nur eine RS232 mit Logikpegel über ein FTDI–Kabel ist, kann man sehr leicht für eine galvanische Trennung sorgen (Abb. 3).

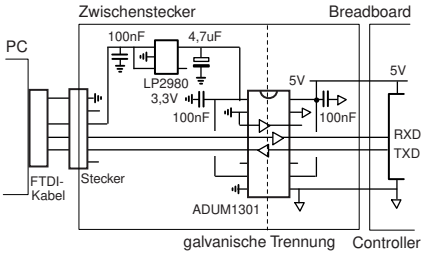


Abbildung 3: Trennung der seriellen Schnittstelle

Damit ist das Oszilloskop das einzige Gerät, bei dem der GND mit dem Breadboard verbunden sein muss. Leider sind moderne Oszilloskope kleine PCs: Wenn sich die Leiterplatte direkt vor dem Flachbildschirm befindet, streut dieser bereits ein. Alternative ist ein altes analoges 60MHz–Hameg–Oszilloskop mit Bildschirm–Röhre für Low–Noise–Messungen im unteren Frequenzbereich.

Oszilloskope mit Lithium–Akkus ermöglichen zumindest zeitweise echten floatenden Betrieb ohne Verbindung zum Schutzleiter. Für eine EMV–Messung bei Burst– und Surge–Pulsen[3] ist das hilfreich.

Masseflächen

Ein drastischer Ansatz, der früher gängiger war, ist eine Metallplatte unter dem Breadboard, die mit dem Schutzleiter verbunden ist. Die kontaktiert über Metallbolzen direkt die Massefläche der Platine.

Abbildung 4: Hack von Jim Williams

Bekannt wurde durch Jim Williams auch der „fliegende Draht–Igel“ über 1–lagiger Leiterplatte (Abb. 4, 16). Pertinax ist dafür günstiger, weil man es leichter sägen kann. Ähnliche Wirkung hat ein geätztes 2–lagiges PCB mit geschlossener Kupferfläche–Oberseite und Leitungen an der Unterseite. Oder für SMD Kupfer unten und Bauteile oben, damit sie leichter zugänglich sind.

[1] EMV steht für ElektroMagnetische Verträglichkeit.

[2] Oder beim Motorrad.

[3] Grundbedeutung: Im Allgemeinen bedeutet „surge“ (engl.) eine plötzliche, starke Bewegung oder einen Ansturm. Das Wort stammt vom lateinischen Verb surgere ab, was „aufsteigen“ oder „sich erheben“ bedeutet. In der Elektronik: Spannungsspitzen.

Probes

Der übliche Ground–Strap[4] für Oszilloskope ist, wie jeder Pigtail[5], nur bei niederen Frequenzen gangbar (Abb. 5). Wenn man z. B. den Ripple eines DCDC–Wandlers messen will, wäre die formale Variante ein echtes Koax–Kabel (Abb. 6), das an beiden Enden terminiert wird. Die 100–Ohm–Last ist für eine Stromversorgung akzeptabel, aber selten für Signalleitungen.

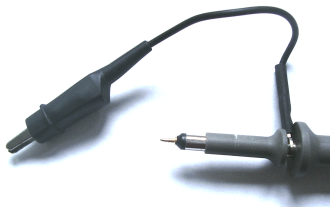


Abbildung 5: Probe eines Oszilloskops

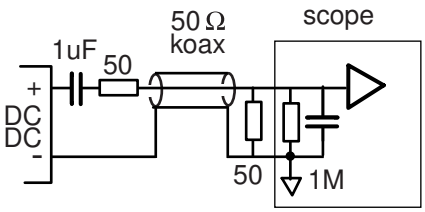


Abbildung 6: 50–Ohm–Kabel

Elektrisch ist es besser, wenn man versucht, die Pins direkt auf den GND an der Hülse des Tastkopfs zu drücken (Abb. 7) [1]. Bei hochohmigen Signalen ergibt sich aber oft ein kapazitiver „Hand–Effekt“, der keine reproduzierbaren Messungen erlaubt.

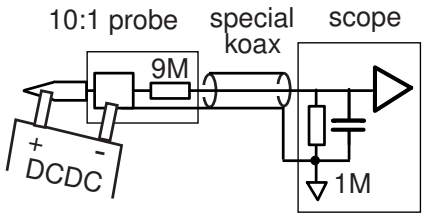


Abbildung 7: Mit Probe, aber ohne Pigtail

Radikale Lösung: kein Kabel [2], sondern ein kleiner PCB–Adapter mit BNC–Stecker, den man direkt ins Scope steckt (Abb. 8). Das ist jedoch nicht populär, denn man kommt nicht mehr an die Knöpfe. Das ist daher nur für sehr kleine Leiterplatten praktikabel.

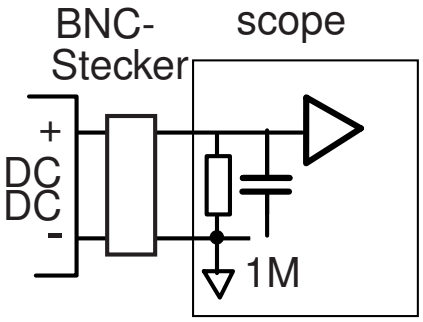


Abbildung 8: Ohne Kabel

Ein Kompromiss ist der kurze Ground–Strap aus Federblech (Abb. 9). Via *ebay.com* aus China billig für unterschiedliche Durchmesser (3,5 ; 4,3 ; 4,5 mm) erhältlich. Testpunkt und Masseflächе frei Hand zu kontaktieren ist aber eher kniffelig. Wenn man große und kleine Kelchkontakte ins Layout integriert, hat man leichter Kontakt (Abb. 10).

Abbildung 9: Feder–Pigtail ...

Abbildung 10: ... mit Buchsen-Kontakten

Für Tektronix[6] gab es schon echte Koax–Buchsen für die Probes (Abb. 11), aber teuer und nicht handelsüblich. Über ebay.com aus China zwar erhältlich (5 mm ; 3,8 mm) (Abb. 12), aber für 8 EUR auch kein Schnäppchen.

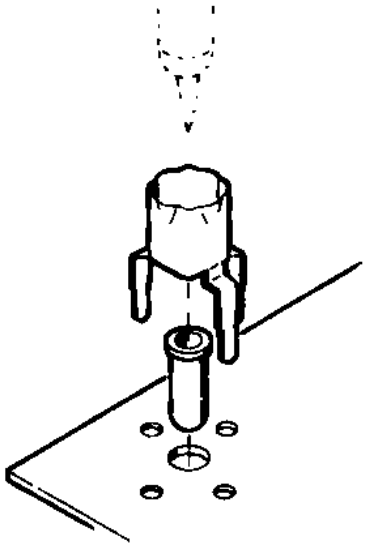


Abbildung 11: Tektronix–Koax–Buchse für Probe 131–5031–00

Abbildung 12: Koax–Buchse China

Buffer

Zusätzliche Schaltungen im Layout können die Anforderungen an die Messtechnik oft reduzieren.

MHz–Takte mit Logikpegel passen nicht für Bananensteckerkabel. Ein

[4] Masseband, auch Erdungsband.

[5] Ein Pigtail (engl. für Schweineschwänzchen) stellt ein Kabel dar, das auf einer Seite mit einem Stecker konfektioniert ist und bei dem die Gegenseite unkonfektioniert ist. Die unkonfektionierte Seite des Kabels kann an eine Kontaktstelle gedrückt werden.

[6] Tektronix ist ein Unternehmen aus den Vereinigten Staaten mit Firmensitz in Beaverton im Bundesstaat Oregon. Es ist insbesondere für seine elektronischen Test– und Ausrüstungsgegenstände wie Oszilloskope, Logikanalysatoren und mobile Testprotokollausrüstungen bekannt.

simpler Teiler reduziert das Signal auf <20 kHz (Abb. 13). Dann ist man im Bereich, in dem Multimeter Frequenz messen können.

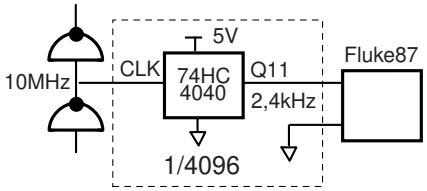


Abbildung 13: Frequenzteiler

Für hochohmige Signale sind auch die 10 MOhm des 10:1–Tastkopfes zu viel Belastung. Ein CMOS–OP ist ideal hochohmig, aber langsamer. Das ergibt als erfreulichen Nebeneffekt jedoch weniger Breitbandrauschen. Beim *TLC271* kann durch Jumper sogar die Geschwindigkeit verändert werden (Abb. 14). Ideal als DIL–IC in einem IC–Sockel aus Einzelkontakten; es wird nur bestückt, wenn das für die Messung nötig ist.

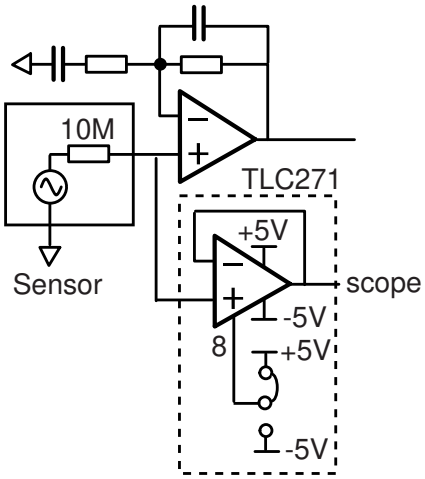


Abbildung 14: CMOS–OP

Zum Messen der Stromaufnahme von Schaltungsteilen kann man einen Shunt vorsehen, der im Normalbetrieb mit einem Jumper überbrückt wird (Abb. 15).

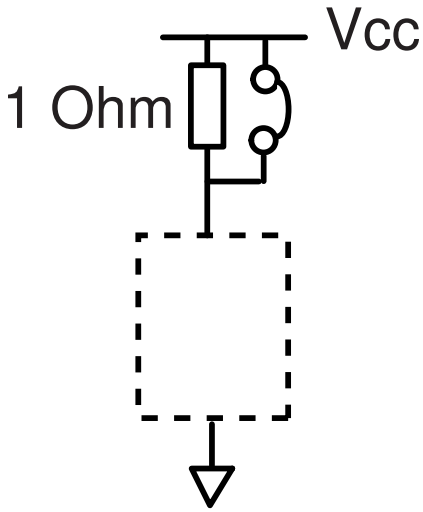


Abbildung 15: Shunt

Quellen

[1] BurrBrown AB162 „DC–to–DC Converter Noise Reduction“

[2] `https://en.wikipedia.org/wiki/Jim_Williams_(analog_designer)`

Abbildung 16: Jim Williams workbench at LTC

Quelle: `https://computerhistory.org/blog/an-analog-life-remembering-jim-williams/`

About NEED and noForth t

Albert Nijhof

The NEED idea allows you to extend a Forth kernel easily and quickly with the extra words you happen to need for the program you are about to write. The LIBRARY is the location where those extra words come from. This can be anywhere, a hard disk, internet, flashrom, etc. A library consists of CHAPTERs. Chapters are independent pieces of Forth code and each chapter has a unique name.

An embedded Forth such as noForth should be as small as possible, but on the other hand, it should be rich enough to be comfortable to work in. Candidates for a place in the library are for example an assembler, a disassembler, words from the *Programming–Tools* word set, some less frequently used words from the *Double–Number* word set and the *String* word set, etc. It is convenient to have such words at hand without them taking up space in forth permanently.

Example 1 — NEED

```
NEED .S
```

This will happen:

- `NEED` searches for `.S` in Forth.
- If `.S` already exists, nothing else is done.
- When `.S` does not yet exist in Forth, the chapter name `.S` is looked up in the library and if found, the `.S` chapter will be loaded.
- When you type `NEED .S` a second time, nothing will happen.

I use the word `LOAD`, but technically it can be `INCLUDE` or `LOAD` or `EVALUATE` or an interpreting function specifically defined for `NEED`.

A chapter is not limited to one definition, it can contain everything understandable to Forth. We can expect that the `.S` chapter contains at least the `.S` definition. It might look like:

```
\ .S
: .S ( -- )
    depth s>d if ." Stack underflow! " exit then
    ." [ "
    s0 cell- swap
    0 ?do cell- dup @ . loop
    drop
    ." ] " ;
```

The noForth t Library

NoForth t (for the RP2040 PICO, ARM) runs in RAM and the RP2040 PICO has quite a bit of flashrom on board. Therefore, the obvious choice is to create a library in flashrom. We could build a dynamic file or block system in flashrom but a static library is much simpler. The drawback is that you can't make changes. You can add chapters or else rebuild the entire library. Our library has a very simple structure, it is just an unordered series of chapters separated by `ctrl-i` (hx 09). A chapter itself contains only ASCII hx 20 . . . 7F with the exception of `cr` (hx 0D) for the line endings.

The first line of a chapter starts with a backslash followed by the name of the chapter. That's all we need to make the chapter findable. `NEED` searches the library for "(09) + backslash + space + name" and a chapter ends when the next (09) is encountered. This library structure is very simple, the Forth code in the chapters has no restrictions.

Two notes:

- noForth communicates via a terminal. Once a `NEED` command is given, its execution no longer uses the terminal. The whole command is executed by noForth, there are *no* terminal timing problems. As a result `NEED` is amazingly fast compared with loading Forth code through the terminal.
- It is rather easy to define forth words concerning the library like
  `CHAPTERS ( -- ) \ display all chapter names`
  `CHAPTER' ( <chapter-name> -- address )`
  `.CHAPTER ( <chapter-name> -- ) \ display chapter contents`

Example 2 — a Nestable NEED

```
NEED [IF]
```

Now the `[IF]` chapter is conditionally loaded. Logically, it will also contain the definition of `[THEN]` because an `[IF]` without a `[THEN]` has no use. Theoretically, we could omit `[ELSE]` from the `[IF]` chapter and create a separate `[ELSE]` chapter with the following content:

```
\ [ELSE]
NEED [IF]
: [ELSE] <forth-code-for-[else]> ; immediate
```

`NEED [IF]` conditionally loads `[IF]` and `[THEN]`. `NEED [ELSE]` conditionally loads `[IF]` `[THEN]` and `[ELSE]`.

You see, `NEED` is nestable.

Forth Style

`NEED` expects a name in the input stream. In general this is practical, but for use within a definition a more forthish style is desireable. `S" NEED name" EVALUATE` is a

solution. Or perhaps `S" name" NEEDED` because `NEEDED` most likely already exists as a factor of `NEED`.

Example 3 — RUN

```
NEED 125MHZ
```

or

```
NEED 250MHZ
```

These chapters contain code to (re)set the processor clock speed. They don't compile definitions named `125MHZ` or `250MHZ`, they are scripts (in this case they don't even compile anything at all). As a result, these chapters are actually *unconditionally* loaded. The word `RUN` is the factor in the `NEED` definition that does the loading, so we can also say `RUN 125MHZ`. `RUN` does not test if a word already exists in Forth, it always loads the chapter.

Example 4 — Assembler

```
NEED ASSEMBLER
```

In noForth this is *not* a good idea because the word `ASSEMBLER` already exists as a *vocabulary* before the assembler is loaded. When you really load the assembler a shield `ASM\` completes the code. Therefore, `NEED ASM\` is a better idea.

Note:

In noForth, `SHIELD` is a variant of `MARKER`. When a shield is executed, it removes all the compiled code behind it, but not the shield itself. For example, `NOFORTH\` is the shield that marks the end of the kernel. Everything in the dictionary behind the kernel is removed when you execute it.

Postscript

Ulrich Hoffmann talks about "NEED Words and Define Source Libs" on YouTube: `https://www.youtube.com/watch?v=kpjlEAV2Ndw`

Willem Ouwerkerk wrote the `NEED` program for noForth t. You can find it on PFW (Project Forth Works). Look for "Library mechanism". `https://wiki.forth-ev.de/doku.php/en:pfw:welcome`

The noForth page: `https://home.hccnet.nl/anij/nof/noforth.html`

(AN 16feb2025)

How to participate is explained here:

`https://project-forth-works.github.io/index-orig.html`

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–Neuostheim

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
microcore (uCore)
Klaus Schleisiek
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
volksFORTH
ultraFORTH
RTX / FG / Super8
KK–FORTH
Ingenieurbüro
Klaus Kohl–Schöpe
Tel.: **(0 82 66)–36 09 862**$_p$

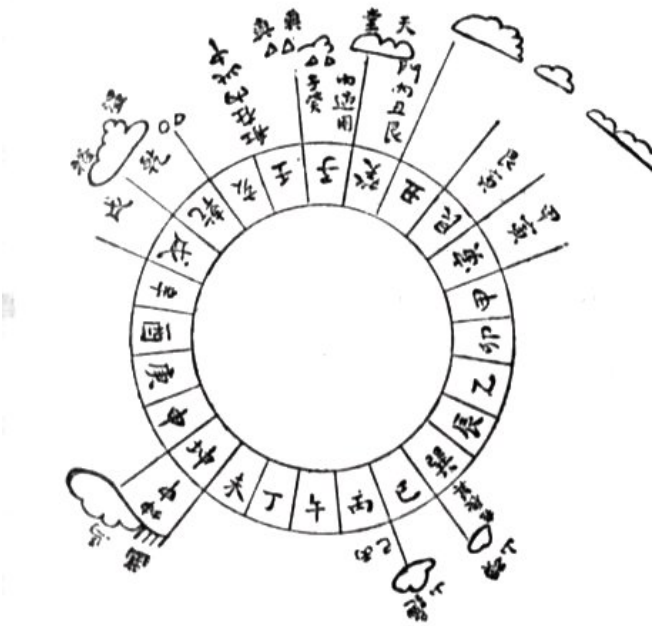
Termine

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 in ungeraden Monaten
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.

Forth–Tagung 2025

24. – 27.04.2025

Das ist das Wochenende nach Ostern, ab Donnerstag.

Jetzt gleich anmelden!

`https://tagung.forth-ev.de/`

So versammeln wir uns im Frühjahr also im Süden Deutschlands. Bernd Paysan hat im schönen Zollernalbkreis in Baden–Württemberg ein Quartier ausgemacht. Schaut es euch schon mal an. Die Preise sind moderat, die Zimmer hübsch, die Tagungsräume sachlich und die Küche ist geräumig — wir werden uns dort selbst verpflegen! Wie in den Anfängen der FG. Zumindest das Frühstück und Abendbrot. Mittags werden wir Essen wohl kommen lassen oder auswärts essen gehen.

Burladingen ist eine Kleinstadt, liegt auf 722 m ü. NN in einer Talspinne der Schwäbischen Alb, an der südlichen Grenze der Mittleren Schwäbischen Alb zur Hohen Schwabenalb.

> „Das *Pleißne* ist ein regionaler Soziolekt aus dem Hausierhandel im Burladinger *Killertal*, der zu den Dialekten des *Rotwelschen* gehört. Bis zur Motorisierung nach dem Zweiten Weltkrieg und dem damit einhergehenden Niedergang des Peitschenhandels diente es den Burladinger Händlern als Geheimsprache. Pleißne, das im Rahmen lokaler Traditionspflege weiter praktiziert wird, hat den Wortschatz der örtlichen Umgangssprache geprägt.“ [Quelle: `https://de.wikipedia.org/wiki/Burladingen#Sprache`]

So passt auch *Forth* ganz gut dahin, da die Ortsansässigen Geheimsprache schätzen.

Abbildung 1: Das WIR–Tagungshaus in Burladingen

`https://www.wir-projekt.de/tagungshaus/`

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

„Die Gesellschaft!“, sagte der kleine Drache.