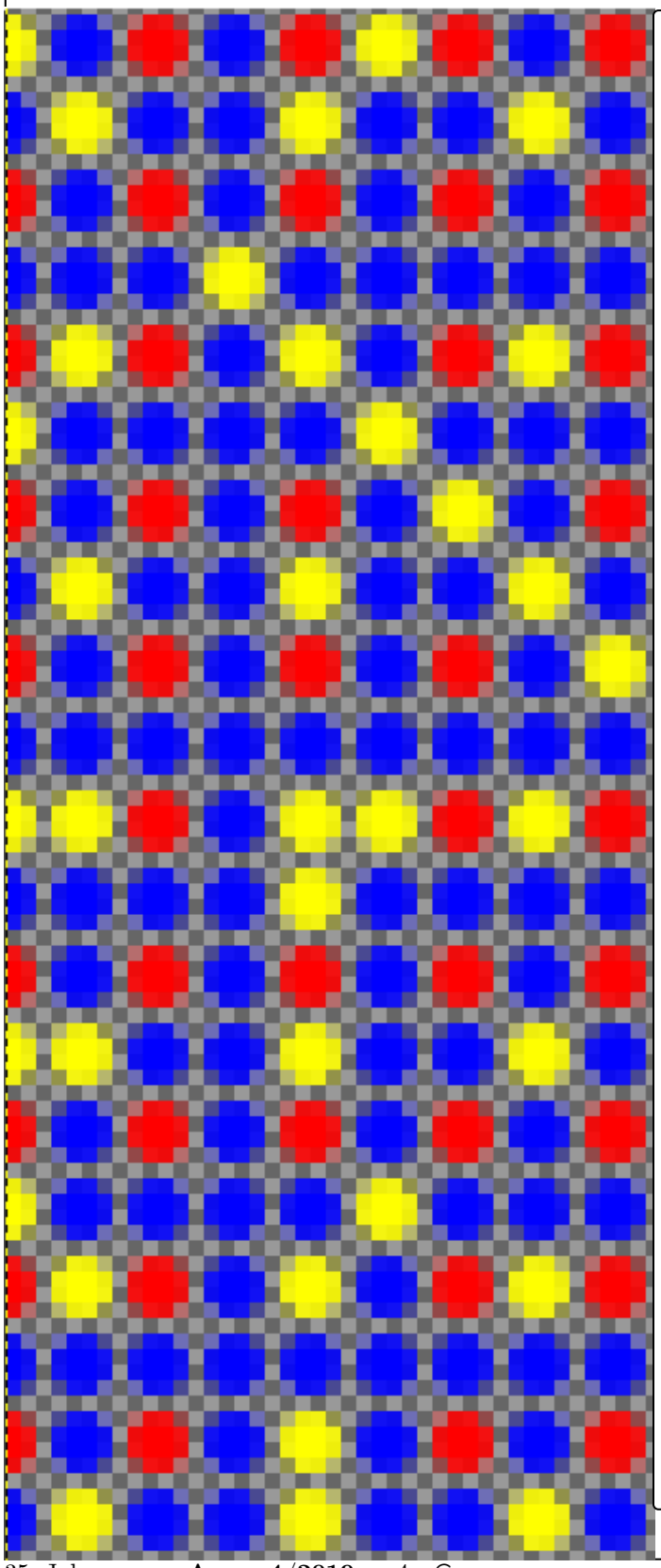
Vierte Dimension

4/2019

Das Forth–Magazin

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



In dieser Ausgabe:

Titelbild: Grafische Darstellung des ggt(x,y), Ausschnittsvergrößerung DE.JBERRIES.COM ... winziger Ausschnitt des Bildes, aus dem Netz gefischt.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: +49(0)6239 9201-85, Fax: -86
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,

kurz vor Weihnachten ist doch noch ein Heft fertig geworden. Und während ich hier tippe, besteht noch Hoffnung, dass es noch unter den Tannenbaum kommt an Heiligabend — früher Modelleisenbahn, heute Forth in MCUs. :-)

Es ist zwar ein bischen dünn ausgefallen, das Heft, aber es gab schon dünnere in den Anfängen unseres Magazins. Liebe Leser, liebe Forthfreunde in aller Welt, nicht nachlassen, tüchtig Forthiges sammeln und herschicken!

Und nun zum Heft. Zustande gekommen ist es wieder durch beharrliches Suchen in allen Ecken der Welt nach Leuten, die sich mit Forth befassen und dann auch was davon hergeben mögen. Als Korrespondenz, als Artikel oder als Kolumne. Ihr braucht keine besonderen Formate zu beachten für euren Text, da machen wir schon was draus. Wir, das sind derzeit Wolfgang Strauß, Bernd Paysan, Ewald Rieger und ich (Michael Kalus), aber das wusstet ihr ja schon, oder? Wolfgang ist der Lektor, Bernd hält die Technik in Gang und unsere Laune hoch und macht den Feinschliff im Satz. Und ich bin der Sammler und Layouter. Ewald schließlich sorgt für den sauberen Druck und Versand der Hefte in alle Welt. Ja, tatsächlich gehen auch Hefte in Papierform über Europas Grenzen hinaus nach Amerika und Asien. Afrika und Australien derzeit (noch?) nicht. Eine Sendung nach China kam nie an. Nach Taiwan jedoch ging es reibungslos.

Was haben wir diesmal im Heft? Kein Leitthema derzeit, gedruckt wurde, was eingetroffen ist. Nicht mehr so mathematiklastig, obschon Jens Storjohann mit *Euklid* anschließt an das letzte Heft. Euklid? Wie war das doch gleich — ggT? Ob Euklid sich vorstellen konnte, in der fernen Zukunft mal von Bedeutung zu sein, so rund 2300 Jahre später? Ich bin sicher, hätte er Forth gekannt, es wäre in seinem berühmtesten Werk *Elemente* benutzt worden.

Dann lässt Manfred Mahlow uns weitere Blicke werfen in seinen Werkzeugkasten. *eForth*, das kleinste Forth, das es gibt und welches er pflegt und gern benutzt, seit es freigegeben worden ist, kann auch dazu gebracht werden, *namespaces* zu handhaben, ebenfalls mit geringem Aufwand. Und er würde sich freuen, euch dabei behilflich zu sein, es einzusetzen.

Das wichtigste Entwicklungswerkzeug in Albert Nijhof's Software–Schmiede ist seine *IDE*. Er folgt dabei der Devise „keep it simple“. Das geht gut in Windows. In Linux hab ich versucht, es ihm nachzumachen, bisher vergebens. Tja, liebe Linuxer, da gibt es noch was zu tun! Obwohl Manfreds *e4thcom* meinem Ideal schon recht nahe kommt. Wenn das nun noch eine Zeichen–für–Zeichen–Kommunikation mit dem Target ermöglichte, wäre ich glücklich.

Und ich selbst war so frei und hab gleich zwei kleinere Beiträge beigesteuert. Einen über *compilierbare Software–Versionsstempel*, zu dem Manfred Mahlow wiederum viel beigetragen hat; danke dafür auch an dieser Stelle. Und einen über den kleinen Kniff, mit Gforth handliche *Sinustabellen* zu machen.

Wir sehen uns dann auf der kommenden Forthtagung in Berlin! Bis dahin euch allen eine gute Zeit, ein frohes Fest und einen guten Rutsch.

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-2019-04

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: Direktorium@Forth-ev.de

Seemacht

Das Schild steht in *St. Ives* an der Südwestküste der ehemaligen Seemacht[1] England und verweist offensichtlich auf ein Bed&Breakfast für Forth–Entwickler am Wasser.

Jörg Plewe

IDE oder NO–IDE, das ist hier die Frage

Forth–Code schreiben und in MCUs laden — klar, aber wie? Mittels Lieblings–Editor und dann Schnipsel mit Copy&Paste in ein dummes Terminal fallen lassen oder den Code per `#include <Schnipsel>` aus einem klügeren Forth-Terminal laden?

Manfred Mahlow zu e4thcom

Am 14.10.19 schrieb Michael:

> „Tja, da haben wir wieder diese zwei Welten … Dass man die nicht zusammenfügen kann, ist wirklich schade.“

Antwort Manfred:

> „Ich vermute, das hat mehr mit Gewohnheiten und Vorlieben zu tun.
>
> Auch mit e4thcom verfolge ich ja so etwas wie einen NO–IDE Ansatz, aber, wie es scheint, mit etwas anderer Schwerpunktsetzung als Albert und Willem.
>
> Ein Editor und gelegentlich ein Dateimanager reichen für eine Entwicklungsumgebung mit e4thcom völlig aus (+ Tool fürs Flashen ;-).
>
> Quelltext wird immer in Dateien geschrieben, auch zum Testen.
>
> Neben Ladebefehlen sind aber auch immer Wörter/Kommandos an der Kommandozeile einzugeben. Die Tipparbeit generell durch Copy und Paste zu reduzieren ist für mich keine optimale Lösung. Mit der Maus zu arbeiten ist für mich auf die Dauer anstrengender als mit dem Keyboard.
>
> Ich habe *e4thcom* (für mich) als Tool für die Programmentwicklung geschrieben. Die Zahl der dabei erforderlichen Tastatureingaben zu reduzieren war und ist dabei eines meiner Ziele. Also editierbare Kommandozeile, aber wichtiger noch eine im Umfang begrenzte aktuelle Historie, einfacher und schneller als die eines Linux–Terminals und durch Anweisungen im Quelltext erweiterbar (siehe \index).
>
> Ein anderer sehr wichtiger Aspekt ist die Faktorisierung von Quelltext in Dateien (Module, Bibliotheken), die bedingt geladen werden können. Da schließt dann auch das während der letzten Forth–Tagung vorgestellte Konzept an, Vokabulare, implementiert als Vokabular–Präfixe, mit Mikrocontrollern zu verwenden.“
>
> `https://forth-ev.de/wiki/events:ft2019:start`
>
> `https://wiki.forth-ev.de/doku.php/en:projects:e4thcom`

mk

An die Autoren: Nehmt LyX

Die Frage war:

> „Wie kann ich für die Vierte Dimension … in TeX Texte verfassen, die einfach eingebunden werden können?“

Nun, am besten macht ihr das *gar nicht* in TeX. Bernd Paysan hat uns vor geraumer Zeit das GUI: “LyX — The Document Processor” verordnet. Und für den gibt es auch ein Template im Repository der FG: `empty-lyx.lyx` heißt es. Damit werden die einzelnen Beiträge erstellt und darin wird korrigiert. Das ist sehr komfortabel, auch um Änderungen nachverfolgen zu können.

Die ganze Umgebung stimmt dann schon mal, die Schriftarten, Überschriften, Listings und Bilder einbinden, Bildunterschriften und Typografisches wie Binde–, Gedanken– und Trennstriche, geschützte Leerzeichen, mathematische Brüche und Formeln setzen, URLs formatieren und all sowas. Kann auch verschiedene Sprachen, sogar chinesische Texte. Und das Beste: Das kann sogar ich handhaben. :)

Über die Steuerdatei `4d2019-04.ltx`, einem Include–File sozusagen, wird aus den einzelnen Beiträgen mittels eines sehr komplexen `make`, das Bernd da pflegt, aus den Lyx–Dateien automatisch erst TeX, die passenden Bilddateien und schießlich aus alledem das Heft. Das Ganze funktioniert natürlich, weil von Bernd, in Linux. Dazu

[1] Achtung Wortspiel: Seaforce :)

musste ich ein modernes Linux installieren, sonst hätte ich da nicht mehr mitspielen können. Hier werkelt also inzwischen ein 64–Bit–Linux–Mint und das geht super damit.

Ihr könnt es aber auch ganz einfach halten und mit einem simplen Texteditor pure Texte erstellen, Plain–ASCII–UTF8, LF oder CRLF am Zeilenende ist auch egal und das so herschicken. Wird dann hier eingebaut. Alle anderen Textprogramme tun es auch, Libre Office, Open Office, Word ... mk

`https://www.lyx.org/`

Übersetzung des Forth–eV–Wiki

Ein Leser im fernen Taiwan bemerkte richtig:

> "... I just happily realized that I can actually read your wiki through simply clicking Google Chrome 'translate to English', why did't I?"

Früher haben wir uns noch selbst die Mühe gemacht, einiges auch in andere Sprachen zu übersetzen. Das ging natürlich sehr schleppend und blieb immer äußerst unvollständig. Wie ihr seht, ist es auch gar nicht mehr nötig heutzutage. Wunderbar!

Und er fuhr fort:

> "And Wow! Your, or our, magazine is the only Forth magazine in the world. That touches. I can't find any donation link, where is it?"

Nun, es gibt einfach keinen Spenden–Link. Aber ihr könnt Forth–Beiträge spenden, soviel ihr wollt! :-)

Um zu zeigen, dass die *Vierte Dimension* dort tatsächlich auch ankommt, sandte er gleich das Beweisfoto mit. Hier ist es:

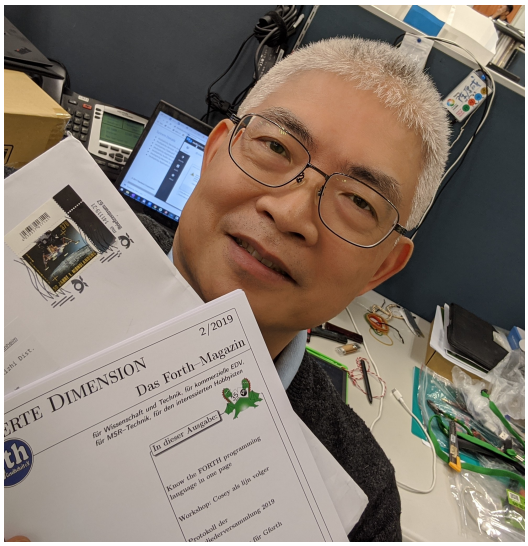


Abbildung 1: Die 4d2019-03, frisch eingetroffen in Taiwan

Quelle: C.H. Chen, Email vom 14. November 2019. mk

Project–k FORTH kernel

> (Dezember 2019) ... "If it's suitable for a printed magazine I'd like to introduce this tiny FORTH kernel and listen to your suggestions: `https://github.com/hcchengithub/project-k/blob/master/notebooks/tutor.ipynb`
>
> Best regards, H.C. Chen"

Was sich dahinter wohl verbirgt? Auf GitHub erfährt man, dass *Project-k* ein sehr kleiner Kernel der Programmiersprache FORTH ist, der Javascript und Python–Open–Sourcing unterstützt.

GitHub `https://github.com/hcchengithub/project-k`

Es soll dieser FORTH–Kern nun verwendet werden, um ein eigenes kleines FORTH–System für verschiedene Rechner zu erstellen — auf virtuellen Maschinen.

> "Use an online Jupyter Notebook, I recommend notebooks.ai while Google colab, Microsoft Azure Notebooks, and more are available out there, you don't need to install anything. Click [Download Zip] from GitHub project-k ..."

Nun, dann sagt mal Bescheid, was ihr da herausgefunden habt. mk

2020: Makerthon statt Makerfaire in Ostwestfalen-Lippe

Im November 2019 emailte Esther Rüßler <esther.ruessler@city2science.de>

> „Liebe Maker,
>
> vielleicht habt Ihr euch schon gefragt, wie es mit dem Kreativ- und Technikfestival MAKE OWL im Jahr 2020 weitergehen wird? Nach dem Erfolg der ersten Maker Faire OWL im Herforder Güterbahnhof planen wir auch weiterhin eine Fortsetzung, allerdings im Wechsel mit einem neuen, spannenden Format: Für das kommende Jahr planen wir den ersten MAKERTHON OWL, bei dem kreative Teams aus Makern, Unternehmen, Hochschulen und Bildungseinrichtungen Lösungen für regionale Herausforderungen entwickeln.
>
> Da sich das MAKE OWL Festival nicht parallel organisieren lässt, werden wir dieses Event nun erst im Sommer 2021 fortsetzen.
>
> Informationen zum geplanten MAKERTHON und anderen Maker–Aktivitäten in OWL erhaltet Ihr auf `www.makeowl.de` oder auf Facebook `https://www.facebook.com/makeowl/`.
>
> Wir freuen uns darauf, in der Zwischenzeit mit Euch an spannenden Themen und Projekten rund um die Maker Kultur in Ostwestfalen–Lippe und weit darüber hinaus zusammenzuarbeiten!
>
> Wir wünschen Euch bereits jetzt eine besinnliche Adventszeit!

Esther und Annette“ mk

Makerfaire in München — Papierflieger, ohne Forth

Aus der Korrespondenz mit Rafael Deliano im November 2019 über Makerfaires:

„‚Maker‘ ist kein ‚Forth e.V.‘–Thema. Es gibt nur eine sehr überschaubare Schnittmenge zwischen Makern einerseits und FORTH auf Mikrocontrollern und Retro–Homecomputern andererseits.

Das Pubkikum auf der Messe erwartet keine Software auf PCs, es ist keine Computermesse. Es ist keine ‚embedded‘–Messe in dem Sinn, dass Mikrocontroller kein aktuelles Thema mehr sind.

Was erwartet wird, sind innovative, bezahlbare Werkzeuge wie 3D–Drucker, CNC–Fräsen ... Arduino und Raspberry–Pi werden dort als Hardware mit integrierter Software zur Erstellung von Applikationen verkauft. FORTH kann da nur Mittel zum Zweck sein. Das widerspricht völlig der Ideologie des Forth e.V.

‚Nachteil‘ einer Messe ist der heftige Konkurrenzkampf bezüglich der Aufmerksamkeit des Publikums. In Erinnerung bleiben spektakuläre Applikationen, die mit DIY–Mitteln realisiert wurden. Es wird toleriert, dass sie meist nicht sinnvoll sind. Auf dieser Schiene laufen auch Nostalgieprodukte à la 8–Bit–Retro–Homecomputer. Das sind die Ursprünge des Raspberry–Pi, der mal als moderner Ersatz für Homecomputer konzipiert war.

Man hat also hohe Hürden, was man ausstellt und wie gut man seinen Stand aufbaut. Viel Publikum bedeutet auch: Typischer ‚Familienausflug‘ von Leuten, die nicht programmieren und nicht löten können.

Es ist zwar keine konventionelle Industrie–, sondern eher eine Freizeitmesse. Auf der typischen Freizeitmesse wurde mal der vorindustrielle *Apple I* ausgestellt. Also, was soll da heute hin? Förderer der Messe ist deshalb u. a. die Landeshauptstadt München, Referat für Arbeit und Wirtschaft, die sie gerne als ‚Woodstock der Maker‘ tituliert.

In der Hoffnung, dass aus dem Dunstfeld der Messe ein *Apple //* kommen wird. Auf einer Maker–Messe als Innovations–Plattform erwartet das Publikum keinen Mainstream. Das wäre ein Vorteil für FORTH.

MfG JRD“

Und was waren Hingucker in München? Rafael hat zwei Papierflieger–Macher herausgepickt. Abb. 2 zeigt Martin Laarmann. Der Ingenieur, der dieses Papierflieger–Maschinengewehr aus einem Akku–Schrauber von Bosch entwickelt hat, macht das als Hobby. Seine Spezialgebiete sind die Avionik und eben Papierflieger. Der Rest des Gerätes stammt aus dem 3D-Drucker. Abb. 3 zeigt Dieter Krone und seine Maschine.

„Hinten ein DIN-A4-Blatt rein, vorn schießt ein Flieger raus.“ (Spiegel Online; Papierflieger: Ein Ingenieur gibt Anleitung zum Falten. 2015)

Er macht das schon länger, wie ihr seht. mk

Abbildung 2: Papierflieger–Maschinengewehr

Abbildung 3: Papierflieger–Maschinenpistole

Der Euklidische Algorithmus, Großvater aller Algorithmen

Jens Storjohann

Der Euklidische Algorithmus gehört nach Meinung des Autors zur Allgemeinbildung für Informatiker und Informatik–Praktiker. Hier wird er beschrieben und es wird gezeigt, wie er durch ein kleines Programm–Schnipselchen, natürlich in Forth, ausgeführt werden kann.

Grundlagen und Bezeichnungen

Beim Umgang mit ganzen Zahlen tritt u. a. die Aufgabe der Division als Umkehrung der Multiplikation auf. Anhand des Beispiels

```
28 : 6 = 4 Rest 4
```

bilden wir die Begriffe *Dividend (28), Divisor (6), Quotient (4)* und *Rest (4)*.

Wenn die Division ohne Rest aufgeht, nennt man den Divisor *Teiler*. Wichtig für den noch zu beschreibenden Euklidischen Algorithmus ist, dass die Differenz zweier Zahlen, die beide den gleichen Teiler haben, wieder diesen Teiler hat. Insbesondere der Rest, den man sich vorstellen kann als durch mehrfaches Subtrahieren entstanden, hat auch diesen gemeinsamen Teiler.

Im obigen Beispiel ist der (größte) gemeinsame Teiler 2.

Was bedeuten `ggT` und `kgV`?

Wer in der Schule Bruchrechnen ohne Taschenrechner gelernt und geübt hat, kennt die Begriffe *größter gemeinsamer Teiler* (`ggT`) und *kleinstes gemeinsames Vielfaches* (`kgV`).[1]

Den größten gemeinsamen Teiler benötigt man, wenn man einen gegebenen Bruch so weit kürzen will, dass Zähler und Nenner minimal werden. Man teilt nämlich Zähler und Nenner durch ihren `ggT`. Eine größere Zahl, die Teilen von Zähler und Nenner ohne Rest gestattet, gibt es nicht. Beispiel:

$$\frac{20}{35} = \frac{20/5}{35/5} = \frac{4}{7}$$

Hier wurde ausgenutzt, dass gilt `ggT(20,35)=5`.

Eine geometrische Anwendung zeigt, dass eine Rechteckfläche mit den Kantenlängen `n` und `m` mit Quadraten der Kantenlänge `ggt(n,m)` gepflastert werden kann, wie in Abb. 1 für `ggT(4,6)=2` illustriert ist. Deswegen kann man mit Quadraten der Kantenlänge 2 eine Rechteckfläche mit den Maßen 4 x 6 pflastern.

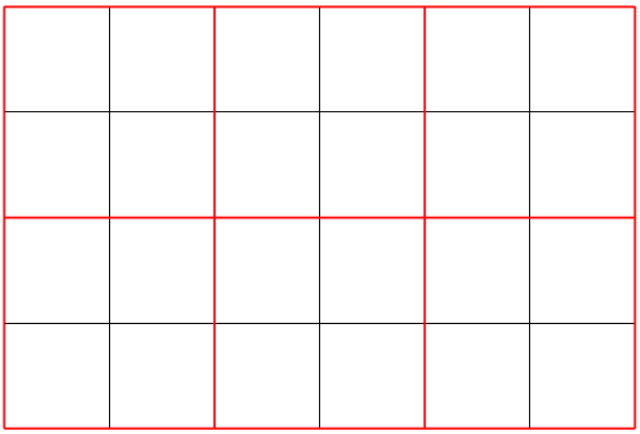


Abbildung 1: Beispiel: Gitter 4 x 6

Das kleinste gemeinsame Vielfache benötigt man, wenn man zwei Brüche addieren und den Nenner der Summe möglichst klein haben will. Ein Beispiel:

$$\frac{3}{4} + \frac{5}{6} = \frac{3*6+5*4}{4*6} = \frac{38}{24}$$

besser:

$$\frac{3}{4} + \frac{5}{6} = \frac{3*3+5*2}{12} = \frac{19}{12}$$

Hier wurde ausgenutzt, dass gilt `kgV(4,6)=12`.

Einige notwendige Formeln

Wir benötigen einige in Formeln gekleidete Gesetze, um das Programm zur Berechnung des `ggT` vollständig zu gestalten, so dass für jede zulässige Eingabe ein richtiges Ergebnis geliefert wird.

```
ggT(a,0)=a
```

```
ggT(a,b)=ggT(b,a)
```

```
ggT(a,a)=a
```

Insbesondere werden alle Argumente durch Anwendung von `abs` positiv oder Null. Die Fälle, dass eine oder zwei Nullen im Argument auftreten, werden abgefangen, weil sie ohne Rechnung gelöst werden.

Der Euklidische Algorithmus

Für zwei positive ganze Zahlen berechnet der Euklidische Algorithmus den `ggT`. Donald Knuth [Knu81] schreibt:

[1] engl.: greatest common divisor (gcd), least common multiple (lcm)

> "[The Euclidean algorithm is] the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."

Der klassische Euklidische Algorithmus `ggT` [Wik19a] berechnet den größten gemeinsamen Teiler, indem er nach einem gemeinsamen „Maß" für die Längen zweier Linien sucht. Dazu wird die kleinere zweier Längen von der größeren abgezogen und dieser Schritt mit dem Ergebnis und der kleineren Länge so lange wiederholt, bis die abgezogene Zahl mit dem Ergebnis identisch ist. Beispiel:

```
143 - 65 = 78
 78 - 65 = 13
 65 - 13 = 52
 52 - 13 = 39
 39 - 13 = 26
 26 - 13 = 13
```

Die modernere Methode ist, eine Division mit Rest durchzuführen.

Ein Beispiel zeigt dies:

```
Zu berechnen ggT(543,701)
701 : 543 =  1 Rest 158
543 : 158 =  3 Rest  69
158 :  69 =  2 Rest  20
 69 :  20 =  2 Rest  19
 20 :  19 =  1 Rest   1
 19 :   1 = 19 Rest   0
```

Also ist 1 der gesuchte `ggT`. Die beiden Zahlen sind also *teilerfremd*. Sie haben nur die 1 als gemeinsamen Teiler.

Das folgende Beispiel gibt einen Eindruck, wie wenig aufwändig der Algorithmus ist, wenn tatsächlich ein echter gemeinsamer Teiler gefunden wird.

```
Zu berechnen ggT(105,147)
147 : 105 = 1 Rest 42
105 :  42 = 2 Rest 21
 42 :  21 = 2 Rest  0
```

Also gilt `ggT(105,147)=21`.

Beim modernen Euklidischen Algorithmus wird in aufeinanderfolgenden Schritten jeweils eine Division mit Rest durchgeführt, wobei der Rest im nächsten Schritt zum neuen Divisor wird. Der Divisor, bei dem sich der Rest 0 ergibt, ist der größte gemeinsame Teiler der Ausgangszahlen. Weiteres Beispiel:

```
3780 : 3528 =  1 Rest 252
3528 :  252 = 14 Rest   0
```

Somit gilt `ggT(3780,3528)=252`.

Das Programm

Weil wir in Forth programmieren, können wir ausnutzen, dass das Teilen ganzer Zahlen mit Gewinnung des Restes standardmäßig durch `mod` zu erhalten ist.

Der wesentliche Teil, *ein Schritt* des Algorithmus, ist gegeben durch

```
: ggT-step ( n1 n2 -- n3 n4 )
  dup >r mod r> swap ;
```

Dies kann man am einfachsten verstehen, indem man mehrere Schritte „mit Papier und Bleistift" ausführt. Die anderen Teile des Programms bringen die Eingangsgrößen auf dem Stapel in eine Standardform: Zwei nicht–negative Zahlen, die Größte zuerst. Wenn eine Eingangsgröße Null ist, gilt die Regel `ggT(a,0)=a` und `ggT(0,0)=0`.

Das kleinste gemeinsame Vielfache `kgV`

Nahe verwandt mit der Berechnung des `ggT` ist die Frage nach dem kleinsten gemeinsamen Vielfachen `kgV`. Wo wird es gebraucht? Beim Addieren von zwei Brüchen mit unterschiedlichen Nennern

$$3/8 + 1/12 = ?$$
$$3/(24/3) + 1/(24/2) =$$
$$(3 * 3 + 1 * 2)/24 = 8/24 = 1/3$$

In der Bruchrechnung gibt der größte gemeinsame Teiler an, wie ein Bruch maximal gekürzt werden kann.

Anwendungsbeispiel

Eine weitere Anwendung des Euklidischen Algorithmus liegt in der Faktorisierung von aus zwei Primzahlen gebildeten Produkten, was in der Kryptographie mit öffentlichem Schlüssel (Public Key) eine große Rolle spielt.

Wenn man eine Zahl `N` als Produkt aus zwei großen Primzahlen `P1` und `P2` erhält, also:

```
N = P1 * P2
```

kann man im RSA–Verfahren damit eine Verschlüsselung erzeugen.

Das „Brechen" dieser Verschlüsselung besteht darin, diese offengelegte Zahl `N` als Produkt aus den nicht offengelegten Primzahlen `P1` und `P2` wiederzugewinnen oder, wie man auch sagt, `N` zu *faktorisieren*.

Wenn man eine andere Zahl gefunden hat, von der man weiß, dass sie mit einer hohen Wahrscheinlichkeit einen gemeinsamen Teiler mit der gesuchten Zahl `N` hat, startet man den Euklidischen Algorithmus und findet dann mit eben genau dieser Wahrscheinlichkeit einen der Faktoren `P1` oder `P2`. Dies kann man unter dem Stichwort „Pollard's Rho–Verfahren" [Wik19c] beschrieben finden.

Ein kürzeres `ggT`–Programm

Nach Fertigstellung dieses Artikels fand ich im Internet unter `https://craftofcoding.wordpress.com/2015/03/23/dont-underestimate-the-power-of-the-forth/` noch dieses Programm:

```
: gcd ( a b -- gcd )
  begin
  ?dup while tuck mod
  repeat ;
```

Es ist elegant und zweifellos kürzer. Es verwendet das Wort `tuck`[2] aus dem CORE–EXT–Wordset und arbeitet für fast alle Arten von Argumenten korrekt. Nur zwei negative Zahlen führen zu einem (falschen) negativen Ergebnis.

Zusamenfassung und Ausblick

Der Euklidische Algorithmus in der hier gezeigten modernen Form, die unmittelbar mit Divisionsresten arbeitet, lässt sich einfach in Forth programmieren.

Wenn man Interesse an diesem Teil der Mathematik hat, kann man Anwendungen, z. B. für die Kettenbruchentwicklung, betrachten. Die Kettenbrüche sind ein Interessengebiet des Autors, so dass ein weiterer Aufsatz für die VD zu erwarten ist.[3]

Ferner gibt es Varianten des Algorithmus, die auf Effektivität in bestimmten Anwendungen getrimmt sind.

Der Euklidische Algorithmus kann auch mit Polynomen durchgeführt werden.

Die Einträge in der Wikipedia für „Euklidischer Algorithmus“ oder für „Größter gemeinsamer Teiler“, die im vorliegenden Aufsatz herangezogen wurden, sind sehr ausführlich [Wik19a][Wik19b].

Wer weitergehendes Interesse an Pollards Rho–Verfahren [Wik19c] hat, wird auch in der Wikipedia fündig.

Listing

```
1    Euklidischer Algorithmus berechnet
2   \ von zwei ganzen Zahlen den
3   \ größten gemeinsamen Teiler ggT
4   \ englisch: gcd
5
6   : ggT-step ( n1 n2 -- n3 n4 )
7     dup >r mod r> swap ;
8   : ggT ( n1 n2 -- n3 )
9     abs swap abs 2dup <
10  if swap then \ Nichtnegativ u Reihenfolge
11    2dup * 0 >
12  if
13    begin
14      ggT-step dup 0=
15    until \ Restebildung ausführen bis Rest Null
16    drop  \ letzter Divisor ist GGT
17  else
18   max
19  then ;
```

Referenzen

[Knu81] Knuth, Donald E.: *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley, 1981. – ISBN 0–201–03822–6

[Wik19a] Wikipedia: *Größter gemeinsamer Teiler — Wikipedia, Die freie Enzyklopädie.* `https://de.wikipedia.org/w/index.php?title=Gr%C3%B6%C3%9Fter_gemeinsamer_Teiler&oldid=192086950`. Version: 2019. – [Online; Stand 15. November 2019]

[Wik19b] Wikipedia: *Kleinstes gemeinsames Vielfaches — Wikipedia, Die freie Enzyklopädie.* `https://de.wikipedia.org/w/index.php?title=Kleinstes_gemeinsames_Vielfaches&oldid=192400524`. Version: 2019. – [Online; Stand 15. November 2019]

[Wik19c] Wikipedia: *Pollard-Rho-Methode — Wikipedia, Die freie Enzyklopädie.* `https://de.wikipedia.org/w/index.php?title=Pollard-Rho-Methode&oldid=192311732`. Version: 2019. – [Online; Stand 15. November 2019]

Was Euklid noch so alles gemacht hat, als „Meister der Dreiecke“ ...

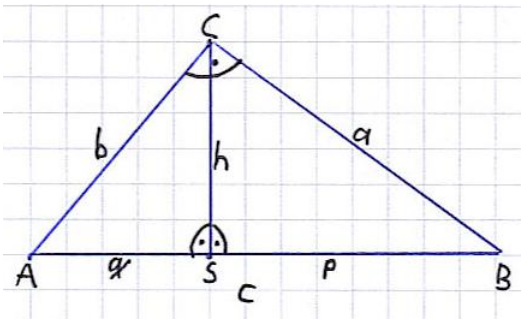


Abbildung 2: Höhensatz des Euklid: $h^2 = qp$

[2] `TUCK ( x1 x2 -- x2 x1 x2 )`
`Copy the first (top) stack item below the second stack item.`

[3] :-)) — (der Sätza)

Namespaces and Context Switching for a Tiny Forth

Manfred Mahlow

Will it be a good idea to add namespaces and context switching to a tiny Forth? Will it then be a tiny Forth anymore? Yes, I think it will. Using namespaces is encouraging concerning factoring of code, not only into words but also into well structured source code modules and implicit context switching makes it easy to define data types or simple objects. Both can make programing much more fun and can help to write readable and well structured code.

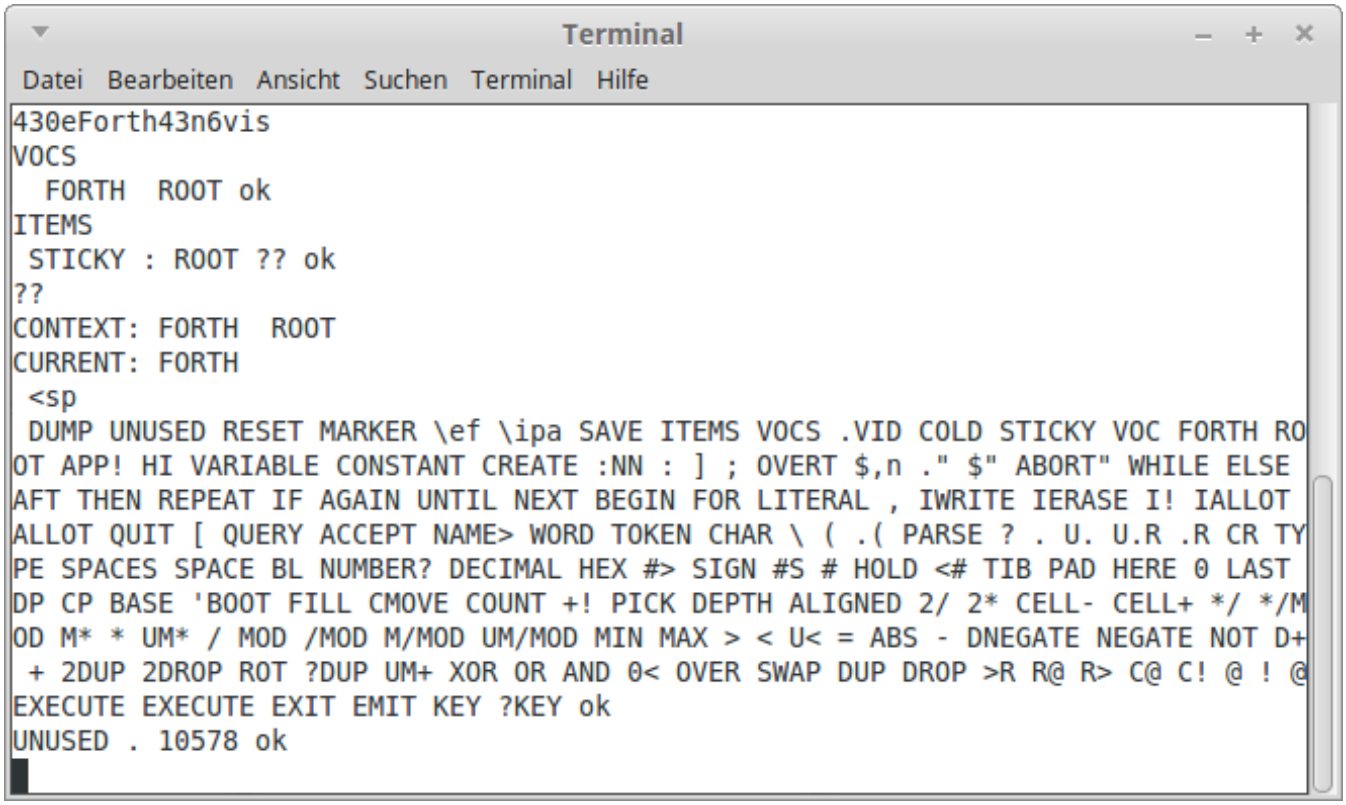


Figure 1: 430eForth43n6vis with all Tools

430eForth43n6vis

This eForth is an extended version of C.H. Tings 430eForth for the MSP430G2553 MCU, based on the naken_ASM version published by M. Kalus on GitHub. [1]

It was created as part of a feasibility study how to extend the support for namespaces in Forth by implementing vocabulary prefixes (`VOC`) instead of F83 vocabularies (`VOCABULARY`). [2]

Wordlists, vocabulary prefixes (`VOCs`), <voc> `ITEMS`, `STICKY` words and related `TOOLS` have been added and two search orders are used, a permanent and a temporary one. [3]

Two Search Orders

The permanent search order is the default search order like the one in Standard Forth Systems. It's restricted here to max. three wordlist identifiers (wid) and it defaults to wid(forth-wordlist) wid(root-wordlist).

The temporary search order is set and activated by vocabulary prefixes.

A vocabulary prefix is a wordlist handler, an immediate word, created with an initially empty wordlist.

Explicit Context Switching

The execution semantics of a vocabulary prefix is to activate the temporary search order with its wordlist identifier (wid) on top.

The temporary search order always holds two wordlist identifiers, the wid of the vocabulary prefix that activated the search order and the wid of the root-wordlist.

An active temporary search order is deactivated after the next word from the input stream is found and compiled or executed.

No F83 VOVABULARYs

The top of an active temporary search order, the wid of the executed vocabulary prefix, can be copied to the permanent search order (see <voc> FIRST , <voc> ALSO). So vocabulary prefixes can fully replace F83 style vocabularies and FORTH and ROOT are then vocabulary prefixes which are also members of the permanent search order.

Implicit Context Switching

By assigning the execution semantics of a VOC as an extra immediate execution semantics to a new word, when it's created, a context switch will be done in interprete or compile mode after the words execution semantics is executed or compiled. May sound complicated but it's easy to use, e.g.:

```
VOC DS1621  DS1621 DEFINITIONS
  ... \ DS1621 methods go here
FORTH DEFINITIONS
DS1621 ITEM 0 CONSTANT TS1 ( -- id )
```

TS1 is an implicit context switching word, a combination of a VOC and a CONSTANT.

In interpret/compile mode TS1 is executed/compiled as CONSTANT and afterwards the DS1621 namespace (the temporary search order) is activated.[1]

The next word, follwing TS1 in the input stream, is then not looked up in the FORTH but in the DS1621 namespace.

Image Size and Tools

All required words to support namespaces and context switching have been added to the 430eForth image, increasing its size from 4.264 to 4.814 of 16.332 bytes.

Extra tools required for flash management, browsing the dictionary and testing can be loaded from source code files, using 940 more bytes.

The image, tools, docs and examples will be made available in the Forth-ev Wiki [4].

User Interface

Any serial terminal program can be used (B9600, 8N1, no handshake) but in the enclosed source code files #include and #require are used, file upload commands of the e4thcom terminal program [5], so it would be the best choice. Start it with the -t 430eforth-xas command line option.

Unfortunately e4thcom is only avaliable for the Linux OS. In a Windows environment you may consider to use PuTTY to connect to an e4thcom terminal on a Raspberry Pi or you can try to run the 64 bit e4thcom in the Windows 10 WSL subsystem [6].

Feedback

What do you think about VOCs and ITEMs ? I'm interested in your feedback.

manfred.mahlow@forth-ev.de

Glossary

Used symbols

I immediate word

D defining word

S sticky word

<voc> a vocabulary prefix

New words in the 430eforth image

<voc> ' ("name" – xt)

Find the word name in the temporary <voc> search order and return its xt.

:NN (– xt)

Start a headerless colon definition and return its xt on the data stack.

[1] The context switching semantics of the VOC is never compiled.

<voc> DEFINITIONS (–) I

Make the wordlist of the vocabulary prefix <voc> the new compilation context.

<voc> FIRST (–) I

Overwrite the top of the permanent search order with the top of the temporary <voc> search order.

FORTH (–) I

A vocabulary prefix, the wordlist handler for the forth-wordlist.

ITEM (–)

A prefix for defining words. A prefixed defining word adds the execution semantics of the last executed vocabulary prefix as an extra immediate execution semantics to the next created word to make it a context switching one. The word is not made state-smart!

ONLY (–) I

Reset the permanent search order to its default, e.g.: `FORTH ROOT`.

ROOT (–) I

A vocabulary prefix, the wordlist handler for the root-wordlist. The root-wordlist holds those words that need to be visible in the permanent and in the temporary search order.

STICKY (–)

A prefix for defining words. A prefixed defining word adds an extra immediate execution semantics to the next created word so that the temporary search order, the word is found in later, is hold, until the next word from the input stream is compiled or executed. The word is not made state-smart!

VOC ("name" –) D

Create a vocabulary prefix with a new empty wordlist. Use name `DEFINITIONS` to make the new wordlist the current compilation context.

.. (–) I

Switch back from a temporary search order to the permanent search order.

Tools from source code files

`#require` and `#include` are e4thcom commands for file uploading.

#require ?? (–) S

Show the current search order, the data stack and all words of the first wordlist in the search order. Does not change the actual search order.

#require DUMP (a –)

Dump 128 bytes from address `a` as hex numbers and as characters.

#require ITEMS (–)

List all context switching `ITEMs` that are defined in the dictionary.

#require MARKER ("name" –)

Create a marker and `SAVE` the current eForth state. Aborts with message "flash?" if there is not enough free flash memory available. A marker, when executed, removes itself and all later defined words from the dictionary and resets the dictionary pointers to their previous values.

SAVE (–)

Save the current eForth state. Words not `SAVEd` are unlinked from the dictionary on `COLD` but the used flash memory is not erased (see `RESET`).

#require RESET (–)

Reset the 430eForth. Resets the user variables `CP DP LAST CONTEXT` and `'BOOT` , deletes all user-defined words and interrupt vectors from Flash Memory and updates the cold-start data in the Flash Information Memory.

#require UNUSED (– u)

Return the size of free dictionary memory in address units (bytes).

#require VOCS (–)

List all `VOCs` that are defined in the dictionary.

.VID (a –)

Given the name address of a word print its name with all <voc> prefixes.

#require TOOLS

Conditionally upload the above-mentioned tools all at once.

Extra words from source code files

#require ALSO (–) I

Copy the top wid from the temporary <voc> search order on top of the permanent search order. Abort with an error message if the permanent search order is full. See also `FIRST` , `ONLY` and `PREVIOUS`

#require PREVIOUS (–) I

Remove the top wid from the permanent search order.

#require COMPO (–)

A prefix to make the next created word compile-only.

#require IMMED (–)

A prefix to make the next created word immadiate.

Referenzen

[1] Michael Kalus, MSP430eForth43n1, GitHub
https://github.com/mikalus/eForth43-msp430g2553-naken

[2] Manfred Mahlow, VOC statt VOCABULARY, Forth-Tagung 2019
https://wiki.forth-ev.de/lib/exe/fetch.php/events:ft2019:voc_statt_vocabulary.pdf

[3] Manfred Mahlow, VOCs ITEMs und STICKY Words, Forth-Tagung 2019
https://wiki.forth-ev.de/lib/exe/fetch.php/events:ft2019:vocs_items_und_sticky_words.pdf

[4] Forth - Namespaces and Context Switching
https://wiki.forth-ev.de/doku.php/projects:forth-namespaces:start

[5] e4thcom - A Terminal for Embedded Forth Systems
https://wiki.forth-ev.de/doku.php/en:projects:e4thcom

[6] Running e4thcom on Windows 10
https://github.com/TG9541/stm8ef/wiki/Running-E4thCom-on-Windows-10

Namenskonflikte ...

... treten im wirklichen Leben ständig auf. Zum Beispiel hat fast jede Schule mindestens zwei Schüler in einer Klasse, die denselben Vornamen haben.

Das verwirrt. Und der Prozess, die gemeinte Person, über die wir sprechen, zu finden, indem wir nach anderen Informationen als einem Vornamen suchen, könnte vermieden werden, wenn jeder einen eindeutigen Namen hätte. Dies ist in einer Klasse von sagen wir mal 30 Schülern kein Problem. Es wird jedoch immer schwieriger, für jedes Kind in einer Schule, einer Stadt, einem Land oder der ganzen Welt einen eindeutigen, aussagekräftigen und auch noch leicht zu merkenden Namen zu finden. Ein weiteres Problem bei der Vergabe eines eindeutigen Namens an jedes Kind besteht darin, dass es sehr anstrengend sein kann, festzustellen, ob jemand anderes sein Kind bereits Macey, Maci oder Macie genannt hat.

Ein sehr ähnlicher Konflikt tritt auch bei der Programmierung auf. Wenn wir ein Programm mit nur 30 Zeilen ohne externe Abhängigkeiten schreiben, ist es sehr einfach, allen Variablen eindeutige und aussagekräftige Namen zu geben. Das Problem tritt auf, wenn ein Programm tausende von Zeilen enthält und auch einige externe Module geladen hat. So sind *namespaces* von Bedeutung und die Bereichsauflösung, für die sie gelten — in Forth wie in anderen Programmiersprachen. mk

e4thcom — Zeitstempel für das Target

Michael Kalus

Neulich tauchte der Wunsch auf, in den Quellcode für ein Projekt mit noForth für eine selbständige Platine einen „Versions–Stempel“ einfügen zu können. Da der Code direkt auf der Platine entwickelt wurde und nicht cross–compiliert im PC, stellte sich die Frage, wie man das Datum und die Zeit des PCs dennoch in das Target compiliert bekommt. Denn sowas ist „nice to have“, weil man ja doch immer vergisst, händisch seine Versionierung vorzunehmen.

Das Ziel war es, so etwas zu haben[1]:

```
: .VER ( -- ) \ Version drucken
  me count type space date+time type ;
```

Auf meine diesbezügliche Mail an den Autor des e4thcom–Terminals, Manfred Mahlow, kam dazu folgende Antwort:

> „Das oder Ähnliches ins e4thcom selbst einzubauen würde bedeuten, dass ich eine wesentlich umfassendere Vorverarbeitung der zum Target zu ladenden Quelltextzeilen implementieren müsste, als das bisher der Fall ist. Das würde wesentlich mehr Code und Rechenleistung erfordern. Deshalb kann ich es mir im Moment nicht so recht vorstellen.
>
> Ich kann aber folgende Lösung anbieten. Im Listing sind drei kleine Dateien, die du in das jeweilige Projektverzeichnis kopieren kannst.
>
> Mit `#i[nclude] date+time` kannst du dann das Wort
>
> date+time (-- ca u)[2]
>
> laden.[3] Das macht, was du gerne hättest, den Zeitstempel.“

Der Trick mit dem Betriebssystem–Aufruf

Mit `#i[nclude] date+time` wird die Datei `date+time` geladen, die folgenden Inhalt hat:

```
\res sh" ./date+time.sh"
#include date+time.fs
```

In der ersten Zeile wird die Datei `date+time.sh` in der *bash* ausgeführt, die Datum und Zeit vom Betriebssystem holt und diese Zeichenkette in der Form eines Forth–Quelltextes in die Datei `date+time.fs` schreibt. Was die compilierbare String–Konstante `date+time` ergibt.

Mit der zweiten Zeile wird dann die soeben erzeugte Datei `date+time.fs` zum Target hochgeladen und compiliert.

Resultat

Dem Forth–Target steht nun das Wort `date+time` zur Verfügung, als Zeichenkette mit dem Datum und der Zeit vom Zeitpunkt, als `#include date+time` ausgeführt worden ist.

Wenn also die ganze Applikation, nachdem sie ausgiebig getestet worden ist, letztlich einmal komplett neu aus dem Quellcode im Target compiliert wird, ist auch der frische Stempel `.ver` dabei.

Hinweis

Je nach Target muss in Zeile 5 von `date+time.fs` das Wort für das Kompilieren eines Strings angepasst werden. Default ist `s"` (`s\"`).

Und natürlich müssen die Rechte der Dateien so sein, dass `bash` das Ganze auch machen darf. Also bitte die Datei `date+time.sh` im Eigenschaftendialog als *ausführbar* markieren.

Listings

Das Shell–Skript `date+time.sh`[4]

```
1  #!/bin/bash
2  #
3  # MM-191116
4  #
5  echo " : date+time ( -- ca u ) s\" $(date)\" ; "
6         > "$PWD/date+time.fs"
7
```

Der davon erzeugte Forth–Quellcode `date+time.fs`

```
1    : date+time ( -- ca u ) s" Sa 16. Nov 17:51:28 CET 2019" ;
```

Die Steuerdatei `date+time.f` , um den Quellcode in die Applikation einzubinden.

```
1  \res sh" ./date+time.sh"
2  #include date+time.fs
```

Anhang

e4thcom -h

[1] Im noForth liefert `me` die Adresse eines *counted string,* in dem die noForth–Version des Kerns abgelegt ist.

[2] ca - compilation address; u - unsigned number

[3] e4thcom -h läd die Hilfe. Darin sind die #-Kommandos erklärt; s. Anhang.

[4] Zeilen 5 & 6 sind im Original *eine* Zeile und wurden hier drucktechnisch umgebrochen.

e4thcom-0.8.0.64 : Serial Terminal for Embedded Forth Systems Copyright (C) 2019 manfred.mahlow@forth-ev.de. This is free software under the conditions of the GNU General Public License with ABSOLUTELY NO WARRANTY.

Usage: /path/to/e4thcom-x.y.z/e4thcom [options]

Options:

-b [B1200,B2400,B4800,B9600,B19200,B38400,B57600,B115200] The baudrate.

Default is B9600.

-d [Device] Serial device to be used.

Default is ttyACM0.

-h Display this help and exit.

-p [dir1:dir2:dir3] Set the path cwd:cwd/dir1:cwd/dir2:cwd/dir3 for Forth source code look-up.

Default is cwd:cwd/mcu:cwd/target:cwd/lib

cwd = current working directory.

-t [<target>] The target to connect to. Loads the target specific plug-in file <target>.efc from the e4thcom-x.y.z directory. 430eforth[-xas] 430camelforth[-xas] 4e4th[-xas] amforth[-xas] mecrisp[-msp430xas] mecrisp-st noforth[-xas] and stm8ef are supported. (xas = with cross-assembler/-disassembler support)

No -t option = simple terminal function, no file upload.

--hdm Enables the half-duplex transmission mode. Default is full-duplex.

--idm Inverse display mode (black or colored letters on white background).

The terminal input is buffered. Editable command line with history and [TAB] selection. The [TAB] selection is based on the first character in the command line buffer. [Shift]+[TAB] clears the command line buffer, [Enter] sends the command line to the target or interprets it as a terminal directive.

Supported terminal directives:

#ls [OPTION]... [FILE]... List project directory content (see ls man page).

#e[dit] [<name>] Edit the file <name>. After an upload error #e[dit] opens the uploaded file.

#i[nclude] <name> Upload the file <name> to the target.

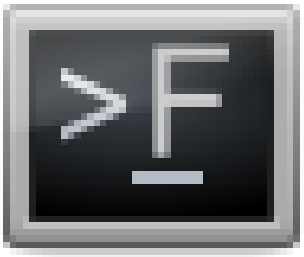
#r[equire] <name> Upload the file <name> to the target, if <name> does not already exist in the (current search order) of targets dictionary.

The default file search order is cwd:cwd/mcu:cwd/target:cwd/lib. It may be changed with the -p option.

The short version of a directive is for the command line only. In source code files the long version must be used, one directive per line. The rest of the line is silently discarded.

Concerning cross assembler, cross disassembler and resource file support please see the e4thcom-x.y.z/doc directory.

Annotation: e4thcom is based on MINFORTH Plus, a MINFORTH derivative.



fsin

Michael Kalus

Neulich wurde für einen Audiotest mit dem MSP430-Launchpad ein Sinuston benötigt. Getestet werden sollte die PWM–Wiedergabe von Audiodaten aus einem Puffer im RAM. Um die Güte der Wiedergabe zu untersuchen, sollte genau eine Sinuswelle im Speicher sein, damit eine definierte und damit bekannte Eingangsgröße mit dem Ausgang am Verstärker verglichen werden kann. Zudem sollten verschiedene Amplituden des Sinus benutzt werden können. Was liegt da näher, als die Daten für den Puffer mit Gforth zu erzeugen.

In Gforth sind eine ganz Reihe von trigonometrischen Funktionen gegeben und operieren mit Gleitkommazahlen. So nimmt `fsin` ein Bogenmaß x vom Stack und gibt den Sinuswert zurück mit 0...1 als Wertebereich. Diesem kann durch die Multiplikation mit der Amplitude die Höhe mitgegeben werden. Führt man diese Operationen für alle x über den Bereich von $2 * pi$ aus, erhält man einen kompletten Datensatz an Sinuswerten für den Bereich.

fdx

Für einen gegebenen Puffer `nx` muss zunächst die Schrittweite `fdx` bestimmt werden zu $2 * pi/n$. Der benutzte Puffer war mit 128 Bytes eher klein. Die Amplitude `amp` durfte sich in einem Byte bewegen, also 0...256 betragen. Da keine negativen Werte vorkommen können bei der PWM–Wiedergabe, sollen die Werte um die Mitte des Amplitudenbereichs liegen — `mw` (Abb. 1).

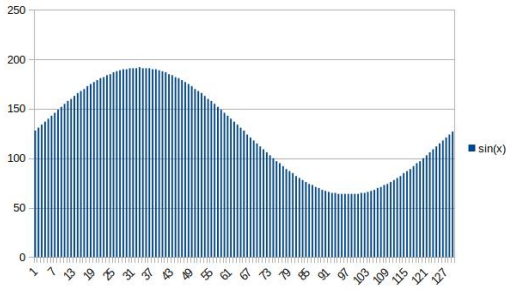


Abbildung 1: Datensatz der Sinuswerte geplottet

fsinx

Durch Gleitkomma–Multiplikation des x–Wertes mit der Schrittweite ergibt sich die jeweilige Stelle für den Stützwert $yi = sin(xi)$. Die Funktion `fsinx` liefert diesen im Gleitkommaformat.

sinx

Durch die Multiplikation mit der Amplitude `amp` und Addition des Mittelwertes `mw` als Bias sowie anschließender Rückführung des Gleitkommawertes in den Festkommawert ergibt sich der gesuchte Byte–Wert.

.tab

Der ganze Datensatz wird sodann in eine Datei gedruckt, fertig formatiert für den Compiler (noForth) der Ziel–MCU. Das ist einfach nur eine Schleife `0...nx` über `sinx`, mit der Compileranweisung `c,` dahinter und der laufenden Nummer als Kommentar zur Beruhigung des Auges des Betrachters. Man sieht dem Forth–Code–Datensatz dann an, dass er vollständig ist. Es ist dem Anwender überlassen, die so gewonnene Datei weiter zu kommentieren, wenn sie in die Anwendung eingebunden wird.

Gforth

Die Anweisung `gforth sin.4th > sinx.txt` erledigt das Ganze elegant. Der Aufruf von `sin.4th` compiliert die Tabelle und druckt sie in die angegebene Zieldatei `sinx.txt`, fertig für den Compiler der MCU.

Viel Vergnügen beim Codieren.

Links

`https://www.gnu.org/software/gforth/`

`http://home.hccnet.nl/anij/nof/noforth.html`

Listing

`sin.4th`

```
1   \ gforth - Sinustabelle in integer Byte-Werten
2
3   : ..s  f.s .s ;
4
5   hex
6   80 value nx  \ Anzahl Stützpunkte
7   80 value mw  \ Mittelwert
8   40 value amp \ Amplitude
9
10
11  : fdx ( -- fdx ) \ Schrittweite im Bogenmaß
12    2 s>f pi f*  nx s>f   f/ ;
13
14  : fsinx ( x -- fsinx ) \ sin(x) (0...1)
15    s>f fdx f* fsin ;
16
17  : sinx ( x -- y ) \ Stützwert (-amp .. mw .. +amp)
18    fsinx amp s>f f*  mw s>f f+ f>s ;
19
20  : .tab ( -- )
21     nx 1+ 0 do
22     i sinx 3 .r space ." c,  "
23     5C emit space i . cr
24     loop ;
25
26   .tab  bye
27
```

NO–IDE

Albert Nijhof

NO-*IDE is the "Integrated" Development Environment for noForth. Compared with other IDEs the amount of type work in* NO–*IDE goes down by 52.01 %, so you can spend more energy on the real programming activity. Research by the distinguished Prof. Z. Prlwytzkofsky shows, that after an accommodation period of two days the productivity of* NO–*IDE users will increase by 105.2 %.*[1]

Components of *NO*–IDE

NO–IDE consists of an MSP430 board with noForth, a simple terminal program on your desktop, an open file in an ASCII–editor, and internet access.

Loading a program

No new commands to learn

Code can be selected from everywhere, from an editor, from an html page, from an email or directly from internet. Copy it and paste it into the terminal. You already know the universal commands:

```
keys:  ^A select all
       ^C copy
       ^V paste
mouse: select
```

No file names to be typed

Avoid typing file names, specially file names with a path. Use the mouse. A filename is to be clicked on, it is typed only once: at its birth.

No "where–am–I?"

You are never in doubt in which "mode" you are. You are where you go with the mouse.

Interactive typing

No `type type type type [enter]` *shit!*

Text longer than 3 or 4 words should be typed in the editor. Do not use the input line as an editor, one doesn't make coffee with a vacuum cleaner. `^C` and `^V` provide that short moment of reflection that obviously wasn't there in `type type type type [enter]` shit! Sounds old–fashioned, but saves you a lot of typing. Of course, short commands like:

```
WORDS
. S
SEE IF
```

go directly into the terminal.

No input gymnastics on a command line

After an error, there is no need to navigate with special commands through previous lines you have typed in the terminal in order to repair something and send that again: the code is before your nose in the editor. You can edit it and then send it, no need for special tricks.

Figure 1: Integrated Development Environment for noForth

Write readable code from the beginning

You could ;) make a habit of writing readable code from the beginning: a stackdiagram after the name of every new word, here and there a hint if necessary and a clear lay–out. (Accumulative programming)

(an 01/04/2018)

Link

`http://home.hccnet.nl/anij/no-ide-gb.pdf`

`https://en.wikipedia.org/wiki/Prlwytzkofsky`

[1] *Of course, this is "gans onwetenschappelijker kwak" of his assistant* Sickbock, *which in no way detracts from the truthfulness of noIDE.*

Forth–Gruppen regional

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.paysan@gmx.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

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`
Treffen alle 1–2 Monate im Unperfekthaus Essen
`http://unperfekthaus.de`.
Termine unter: `https://meetup.com/de-DE/Essen-Forth-Meetup`

Gruppengründungen, Kontakte

Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

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 75 45) – 94 97 59 3**$_p$ `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**$_p$

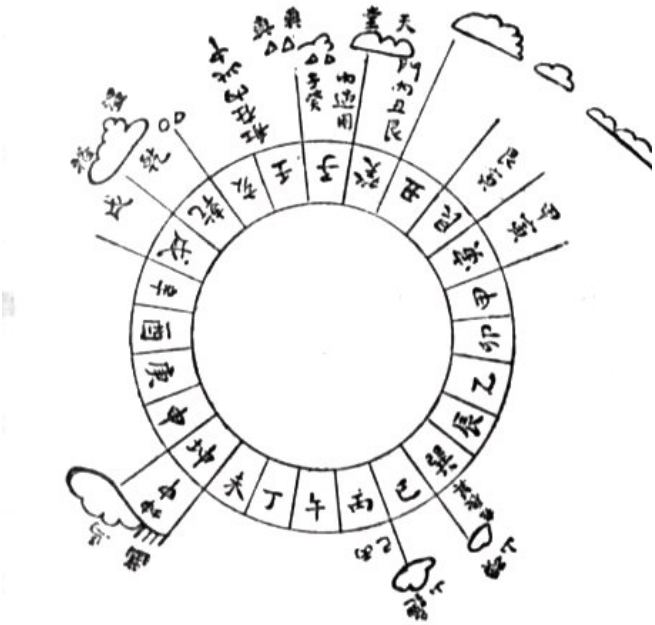
Termine

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*`

27.–30.12.2019
36C3 in Leipzig
`https://events.ccc.de/2019/10/04/36c3-in-leipzig/`

26.–29.03.2020
Forth–Tagung in Klein–Glien bei Bad Belzig, Berlin
`https://tagung.forth-ev.de`

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 in Klein–Glien (bei Bad Belzig)

Carsten Strotmann

Die Tagung im Jahr 2020 führt uns vom 26. bis 29. März 2020 in das CocoNAT Workation Reatreat *in Klein–Glien bei Bad Belzig (ca. 1 Stunde südlich von Berlin). Das CocoNAT — Klein Glien 25 — 14806 Bad Belzig — ist ein umgebauter Gutshof und liegt am internationalen Kunstwanderweg „Hoher Fläming". In der Umgebung gibt es viel Natur und Kultur zu entdecken und Berlin ist nicht weit.*

Anmeldung:

Ab Anfang Januar 2020 wird die Anmeldung unter `https://tagung.forth-ev.de` möglich sein.
Informationen über die Tagungsstätte CocoNAT:
`http://coconat-space.com/de/`