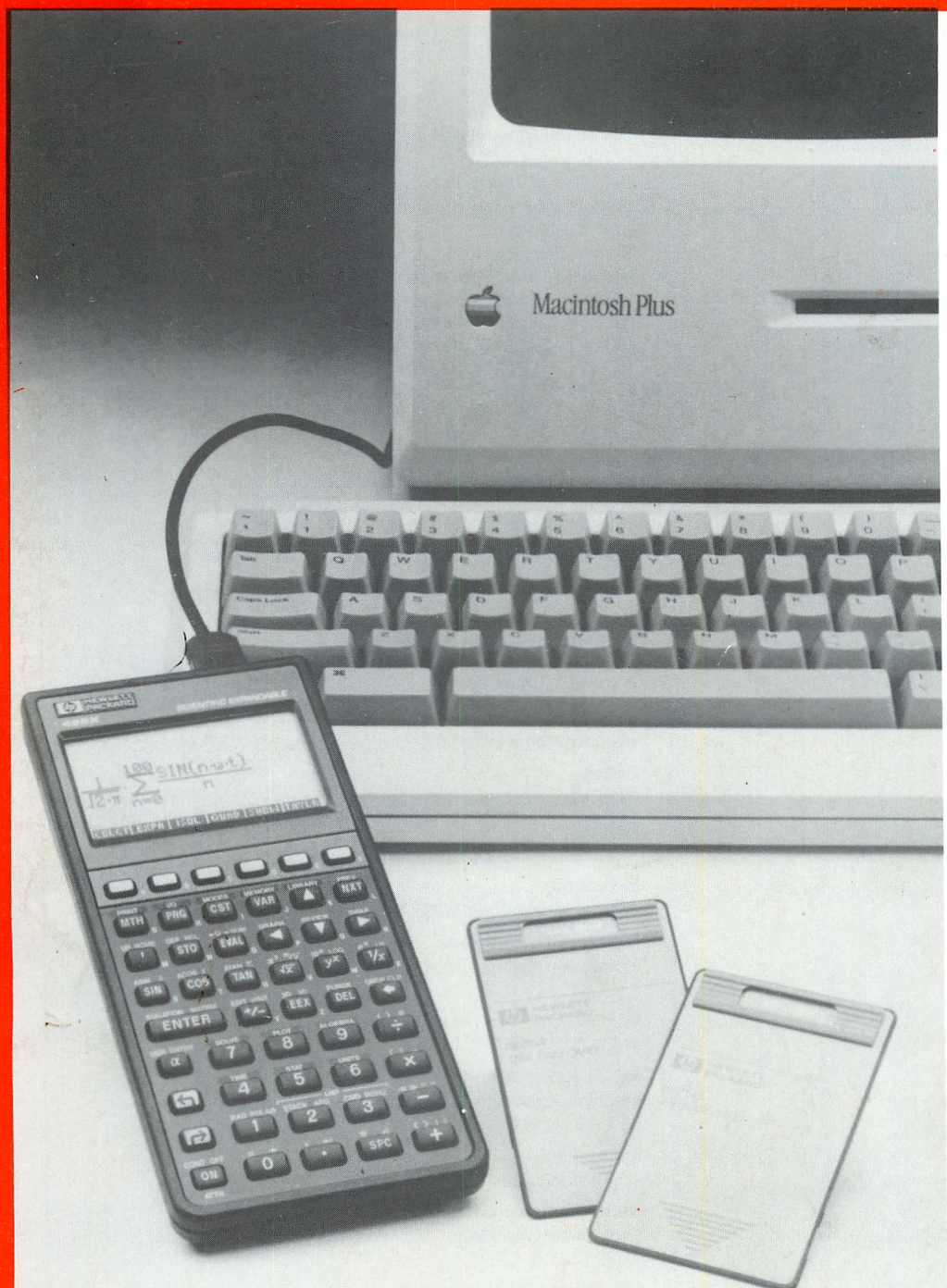
PRISMA

Computerclub Deutschland e.V. • Postfach 11 04 11 • D-6000 Frankfurt am Main 1

Januar/Februar 1991 Nr. 1 D 2856 F



SERVICELEISTUNGEN

BEST OF PRISMA
Schutzgebühr: 30,- DM

Nachsendedienst PRISMA
Schutzgebühr: 5,- DM pro Heft für Jahrgänge 1982-86
10,- DM pro Heft für Jahrgänge ab 1987

Inhaltsverzeichnis PRISMA
Schutzgebühr: 3,- DM in Briefmarken

Programmbiliothek HP-71
Die bislang in PRISMA erschienenen Programme können durch Einsenden eines geeigneten Datenträgers (3,5" Diskette, Digitalkassette oder Magnetkarte) und eines SAFU angefordert werden.

MS-DOS Inhaltsverzeichnis
Kann durch das Einsenden einer formatierten 360 kB oder 1,2 MB 5,25"-Diskette oder einer formatierten 720 kB oder 1,44 MB 3,5"-Diskette und einem SAFU angefordert werden.

ATARI Inhaltsverzeichnis
Kann durch das Einsenden einer 3,5"-Diskette + SAFU bei Werner Müller angefordert werden.

UPLE
Das UPLE-Verzeichnis mit der Kurzbeschreibung der einzelnen Programme sowie den Bezugsbedingungen kann gegen Einsenden von DM 10,- in Briefmarken angefordert werden.

Programme aus BEST OF PRISMA

a) Eine Kopie der Programme von BEST OF PRISMA auf Kassette erfordert das Beilegen einer Leerkassette und eines SAFU.

b) Für Barcodes von BEST OF PRISMA-Programmen gibt es folgendes Verfahren:
Schickt eine Liste mit den Namen und der Seitenangabe (der Barcodeseiten) an die Clubadresse, pro Barcodeseite legt bitte 40 Pf., plus 2,40 DM für das Verschicken, in Briefmarken bei. Die Liste der verfügbaren Programme ist in Heft 3/88 auf der Seite 35 abgedruckt, sie kann gegen einen SAFU angefordert werden.

Der Bezug sämtlicher Clubleistungen erfolgt über die Clubadresse, soweit dies nicht anders angegeben ist, oder telefonisch bei Dieter Wolf:

(069) 76 59 12

Die eventuell anfallenden Unkostenbeiträge können als Verrechnungsscheck beigelegt werden, Bargeld ist aus Sicherheitsgründen nicht zu empfehlen; ist dies nicht der Fall, so wird Rechnung gestellt, dies macht die Sache natürlich nicht unbedingt einfacher, bzw. schneller.
Formvorschriften für Schreiben an die Clubadresse gibt es keine; das Schreiben kann durchaus handschriftlich verfasst sein, ein normaler Sterblicher sollte es noch lesen können. Vor allem den Absender und die Mitgliedsnummer deutlich schreiben !
(SAFU = Selbst Adressierter Freiumschlag)

CLUBADRESSEN

1. Vorsitzender
Gerhard Link (3107),
Postfach 1615, 6090 Rüsselsheim,
✆ (06142) 81 51 0, Fax: (06142) 81 57 9, GEO1:G.LINK

2. Vorsitzender
Alf-Norman Tietze (1909),
Sossenheimer Mühlgasse 10,
6000 Frankfurt 80, ✆ (069) 34 62 40, GEO1:A.N.TIETZE

Schatzmeister
Dieter Wolf (1734),
Pützerstraße 29, 6000 Frankfurt 90,
✆ (069) 76 59 12, GEO1:D.WOLF

1. Beisitzer
Norbert Resch (2739),
Tsingtauerstraße 69, 8000 München 82

2. Beisitzer
Werner Dworak (607),
Allewind 51, 7900 Ulm,
✆ (07304) 32 74, GEO1:W.DWORAK

MS-DOS Service / Beirat
Alexander Wolf (3303),
Pützerstraße 29, 6000 Frankfurt 90,
✆ (069) 76 59 12

ATARI Service / Beirat
Dr. Werner Müller (1865),
Schallstraße 6, 5000 Köln 41,
✆ (0221) 40 23 55, MBK1:W.MUELLER

Regionalgruppe Berlin
Jörg Warmuth (79), Wartburgstraße 17, 1000 Berlin 62

Regionalgruppe Hamburg
Alfred Czaya (2225), An der Bahn 1, 2061 Sülfeld,
✆ (040) 43 36 68 (Mo.-Do. abends)
Horst Ziegler (1361), Schüslerweg 18b, 2100 Hamburg 90,
✆ (040) 79 05 67 2

Regionalgruppe Karlsruhe / Beirat
Stefan Schwall (1695), Rappenwörthstraße 42,
7500 Karlsruhe 21, ✆ (0721) 57 67 56, GEO1:S.SCHWALL

Regionalgruppe Rheinland/Ruhrgebiet
Jochen Haas (2874), Roßstraße 27, 5000 Köln 30,
✆ (0221) 51 98 70

Regionalgruppe München / Beirat
Victor Lecoq (2246), Seumestraße 8, 8000 München 70,
✆ (089) 78 93 79

Regionalgruppe Rhein-Main
Andreas Eschmann (2289), Lahnstraße 2, 6906 Raunheim,
✆ (06142) 46 64 2

Beirat
Manfred Hammer (2742), Oranienstraße 42, 6200 Wiesbaden

Beirat
Peter Kemmerling (2466), Danziger Straße 17, 4030 Ratingen

Beirat
Martin Meyer (1000), Kelkheimer Straße 20, 6232 Bad Soden 1

E-Technik
Werner Meschede (2670), Sorpestr. 4, 5788 Siedlingshausen

Grabau GR7 Interface
Holger von Stillfried (2641), Am Langdiek 13, 2000 Hamburg 61

Hardware 41
Winfried Maschke (413), Ursulakloster 4, 5000 Köln 1,
✆ (0221) 13 12 97

HP-71 Assembler (LEX-Files)
Matthias Rabe (2062), Teichsheide 13, 4800 Bielefeld,
GEO1:M.RABE

Mathematik
Andreas Wolpers (349), Steinstraße 15, 7500 Karlsruhe

Naturwissenschaften
Thor Gehrmann (3423), Hobeuken 18, 4322 Spockhövel 2,
✆ (02339) 39 63

Programmbibliothek HP-71
Henry Schimmer (786), Homburger Landstr. 63,
6000 Frankfurt 50

"Clubadresse"
CCD e.V., Postf. 11 04 11, 6000 Frankfurt 1, ✆ (069) 76 59 12

Einladung zur Mitgliederversammlung 1991

Der Vorstand des Computerclubs Deutschland e.V. lädt ein zur Mitgliederversammlung 1991.

Ort: Frankfurt am Main, Intercity-Restaurant im Hauptbahnhof

Zeit: Samstag, den 7. Dezember 1991, 11:00 Uhr

Tagesordnung:

1. Begrüßung durch den Vorstand
2. Feststellung der Beschlußfähigkeit und andere Formalitäten
3. Bericht des Vorstandes
4. Bericht des Beirats
5. Bericht der Kassenprüfer
6. Entlastung des Vorstandes
7. Neuwahl des Beirates
8. Haushaltsplan 1991
9. PRISMA
10. Anträge
    a) Antrag auf Satzungsänderung
    b) Antrag betreffend Mitgliederversammlung 1992
    c) sonstige Anträge
11. Verschiedenes

Auf Grund der Beschlußunfähigkeit der ordentlichen Mitgliederversammlung vom Samstag, dem 6. April 1991 mangels ausreichender Teilnahme ist die Mitgliederversammlung, zu welcher nunmehr eingeladen wird, gemäß § 14 Absatz 3 der Satzung des CCD e.V. ohne Rücksicht auf die Zahl der erschienenen Mitglieder beschlußfähig.

Der Vorstand wird zum Tagesordnungspunkt 10 a einen Vorschlag unterbreiten, wonach eine Mitgliederversammlung, die nach einer beschlußunfähigen Mitgliederversammlung stattfindet, am selben Tage stattfinden kann wie die beschlußunfähige Versammlung

Der Vorstand wird zum Tagesordnungspunkt 10 b vorschlagen, wegen zeitlicher Nähe zur Versammlung am 7. Dezember 1991 die Mitgliederversammlung 1992 ausfallen zu lassen.

Für den Vorstand:

Gerhard Link 1. Vorsitzender

Protokoll der Mitgliederversammlung des Computerclubs Deutschland e.V. vom 6.4.1991

Ort: Frankfurt am Main, Intercity-Restaurant im Hauptbahnhof

Beginn: 12:00 Uhr

Tagesordnung gemäß Einladung:

1. Begrüßung durch den Vorstand
2. Feststellung der Beschlußfähigkeit und andere Formalitäten
3. Bericht des Vorstandes
4. Bericht des Beirats
5. Bericht der Kassenprüfer
6. Entlastung des Vorstandes
7. Neuwahl des Beirates
8. Haushaltsplan 1991
9. PRISMA
10. Anträge
11. Verschiedenes

Tagesordnungspunkt 1:

Der 1. Vorsitzende Gerhard Link eröffnete die Versammlung, übernahm die Versammlungsleitung und begrüßte die anwesenden Mitglieder im Namen des Vorstandes.

Tagesordnungspunkt 2:

Der Versammlungsleiter teilte mit, daß laut Teilnehmerliste 27 Mitglieder anwesend waren und daß gemäß § 14 Absatz 3 der Satzung des CCD e.V. für die Beschlußfähigkeit die Anwesenheit von mindestens 30 Mitgliedern erforderlich ist. Auch auf Nachfrage meldete sich kein anwesendes Mitglied, welches sich noch nicht in die Teil- nehmerliste eingetragen hatte. Daraufhin stellte der Versammlungsleiter die Beschlußunfähigkeit der Versammlung fest. Nach kurzer Diskussion über die Möglichkeiten einer Satzungsänderung, mit welcher eine solche Situation zukünftig vermieden werden kann, wurde die Versammlung wieder geschlosen.

Ende: 12:10 Uhr.

Liebe Mitglieder des CCD,

aus der obigen Einladung und dem Protokoll der Mitgliederversammlung vom 6. 4. 1991 könnt Ihr entnehmen, daß die Mitgliederversammlung mangels ausreichender Teilnahme nicht beschlußfähig war und daher kurz nach Eröffnung wieder beendet wurde. Nach unserer Satzung hat dies zur Konsequenz, daß zu einer neuen, in jedem Fall beschlußfähigen Versammlung neu eingeladen werden muß. Da die Abhaltung einer Mitgliederversammlung für einen bundesweiten Verein wie den unsrigen nicht nur mit Kosten für den Verein verbunden ist, sondern auch die einzelnen Mitglieder von weither anreisen müssen, hat der Vorstand den Vorschlag des Justitiars aufgegriffen und wird vorschlagen, die Mitgliederversammlung 1991 und die Mitgliederversammlung 1992 sozusagen zusammenzulegen und am 7. Dezember 1991 stattfinden zu lassen.

Nun war es an dem Termin der Versammlung so, daß der übliche rege Gedanken- und Erfahrungsaustausch nach dem offiziellen Schluß der Versammlung stattfand und so keiner der Anwesenden das Gefühl haben mußte, umsonst gekommen zu sein.

Gleichwohl ist die geringe Teilnahme an der Mitgliederversammlung zu bedauern. Gewiß, die stürmischen Zeiten des Vereins, in denen wegen der ungewissen Zukunft des Vereines sehr viele Mitglieder zu den Versammlungen anreisten, sind vorbei. Der Verein bedindet sich im ruhigeren Fahrwasser und erfüllt seine Zwecke offenbar so, daß die Mitglieder mehr oder weniger zufrieden sind und keine Notwendigkeit sehen, dem Vorstand "Dampf zu machen"; daß es sich so verhält, daß die Mitglieder sich resigniert sagen, es sei sowieso Hopfen und Malz verloren, will ich nämlich nicht hoffen.

Trotzdem ist die Mitgliederversammlung keine leere Formalität oder Vereinsmeierei. Je mehr Teilnehmer anwesend sind, desto besser funktioniert der Gedankenaustausch auch zwischen den Mitgliedern und dem Vorstand, desto besser können Vorschläge nicht nur an Vorstand, Beirat, Redaktion usw. herangetragen, sondern gleich innerhalb der Interessierten diskutiert und natürlich auch beschlossen werden.

Nicht zuletzt aber dürfen auch die Vorstandsmitglieder erwarten, daß ihnen für ihre Tätigkeit Entlastung erteilt wird, womit bestimmte rechtliche Folgen verbunden sind.

In diesem Sinne hoffe ich, daß an der ersatzweisen Versammlung am 7. Dezember größeres Interesse besteht als an derjenigen vom 6. April und daß der Vorstand, auch wenn es für die Formalie der Beschlußfähigkeit dann nicht mehr darauf ankommen wird, mehr Anwesende begrüßen kann als beim letzten Mal.

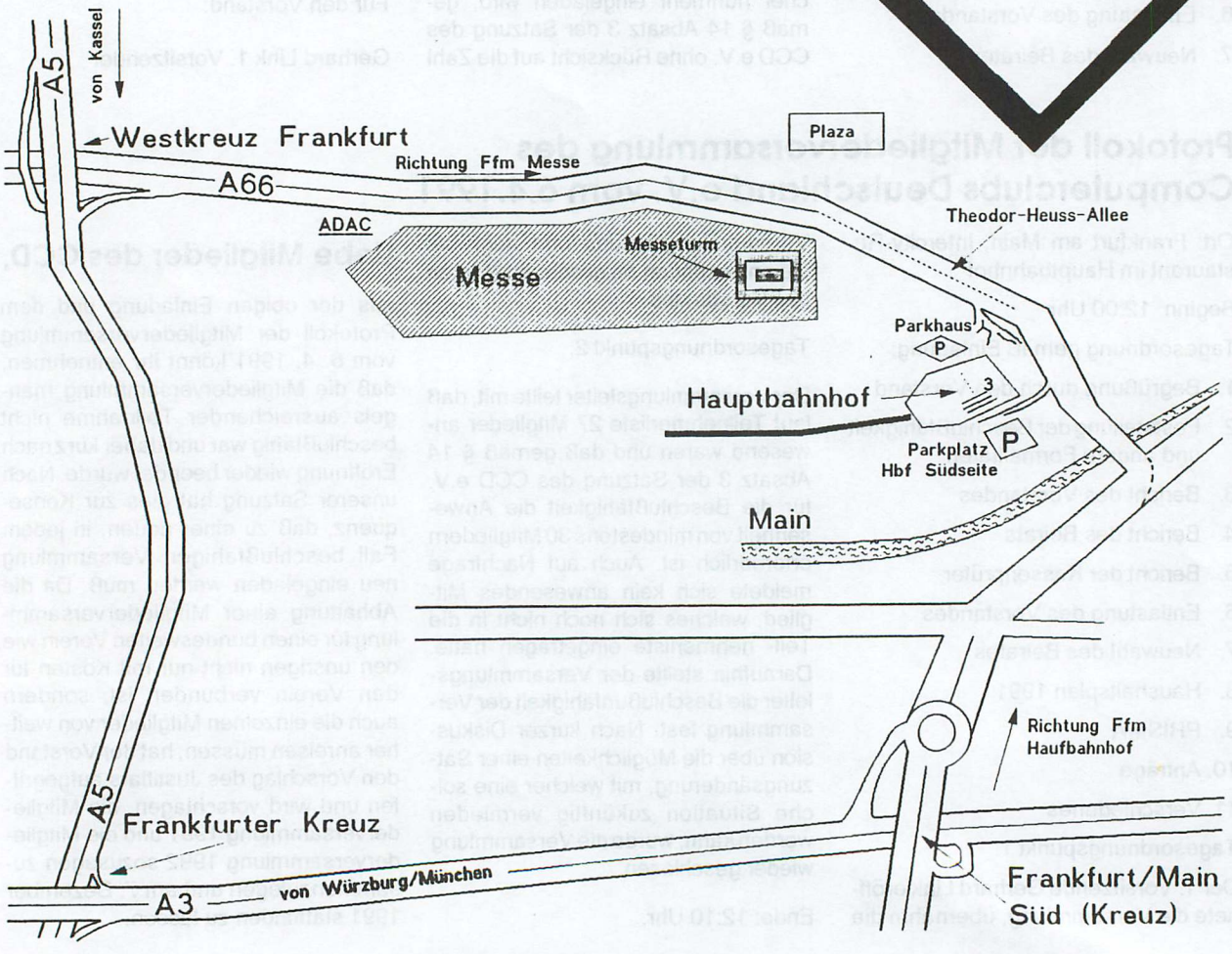
Gerhard Link 1. Vorsitzender

Wie komme ich hin?

Die Anfahrt zum Hauptbahnhof mit öffentlichen Verkehrsmitteln bedarf sicherlich keiner Erklärung.

Im Hauptbahnhof selbst ist der Eingang zum **Intercity-Restaurant gegenüber Gleis 3/4**, der **"Frankfurter Saal"** im **1. Obergeschoß** zu finden.

Für die Autofahrer soll die untenstehende Skizze die Orientierung erleichtern.



Inhalt

CeBIT '91 ohne Besonderheiten

Die diesjährige CeBIT in Hannover bot keine sichtbaren Besonderheiten oder Attraktionen. Bei der PC Hardware galt wie jedes Jahr das Motto "größer - schneller - bunter", wobei sich hier das Attribut "größer" auf das RAM (den Arbeitsspeicher) bezieht. Mittlerweile bietet auch der letzte No-Name-Hersteller einen 386er mit 33 Mhz Taktfrequenz an. Und immer häufiger steht ein 486er an der Spitze der Produktpalette. Das Angebot ist unüberschaubar geworden.

Weil die PC's auf dem Papier fast alle die gleichen technischen Daten aufweisen, unterscheiden sie sich deshalb nur noch durch den Preis? Nein, natürlich nicht, sondern auch in der Verarbeitungsqualität und - man höre und staune - im Kundendienst. Die Produkte der Billig-Hersteller bzw. Handelsketten haben einen technischen Kundendienst am häufigsten nötig. Geboten wird aber oftmals ein miserabler Kundendienst oder sogar überhaupt keiner. Obwohl man doch sogenannte "HighTech"-Computer verkauft, ist das Qualifikationsniveau der Mitarbeiter im Verkauf und im Kundendienst eher als "LowLevel" einzustufen. Bei den Firmen, die dagegen einen qualifizierten Kundendienst bieten, müssen die Kunden diesen erstaunlicherweise nur selten in Anspruch nehmen. Rückschlüsse von der Mitarbeiterqualifikation auf die Produktqualität sind also durchaus zulässig. Unqualifiziertes Personal bedeutet nämlich auch "billiges" Personal - und dort begründet sich zusammen mit der Verarbeitungsqualität der besagte Preisunterschied. Aber Ausnahmen gibt es auch hier.

Reelle Fortschritte sind dagegen eher bei Speichermedien wie Festplatten, CD-ROMs und Tape-Streamern (Bandgeräte) zu vermerken. Eine moderne 60 MB Festplatte (2,5 Zoll Laufwerk) benötigt noch nicht einmal den Platz einer Zigarettenschachtel. Diese fortschreitende Miniaturisierung ermöglicht immer leistungsfähigere Laptop- und Notebook-Computer: 386er SX mit 60 oder 120 MB Festplatte und VGA-LCD entwickeln sich bereits zum Standard. Die ersten Farb-LCDs sind auch schon für teures Geld zu haben.

Im Softwarebereich hat das große Rennen für Anwendungsprogramme unter WINDOWS begonnen. Das 32-te DataTextMaker-DrawBaseManager-Programm mit noch mehr und noch tolleren Bildschirmfenstern ist angekündigt und benötigt mindestens 8 MByte RAM oder am besten gleich 16 MByte für "optimale Perfomance".

Konkurrenzlos !

Weiterhin ohne Konkurrenz bleibt auch der HP48SX. Die asiatischen BASIC-Rechnerchen überfluten zwar den Markt aufgrund ihrer billigen Preise, können aber nach wie vor nicht mit den Möglichkeiten des technisch-wissenschaftlichen Top-Modells von HP mithalten. Der Haken an der Sache ist nur: Von den potentiellen Käufern eines Taschen-Computers wissen dies nur die wenigsten.

Alf-Norman Tietze
(Chefredakteur)

Artikel selbst gemacht

- eine Bauanleitung -

Alle Jahre wieder möchte ich mich an alle diejenigen wenden, die an einer aktiven Gestaltung des CCD mit seiner Clubzeitschrift PRISMA interessiert sind und ihr Wissen einer breiteren Öffentlichkeit zur Verfügung stellen möchten, getreu der Devise des CCD "...zur Förderung von Wissenschaft und Forschung...".
Ich werde immer wieder von Clubmitgliedern gefragt: "Wie muß denn eigentlich ein Artikel aussehen, wenn ich etwas im PRISMA veröffentlichen will". Eine leider oft gehörte These ist auch:
"..ich kann das doch gar nicht...".

Um es kurz auf den Punkt zu bringen: Jeder kann sein Wissen, egal ob Dipl. Ing. oder Student, Professor oder Techniker über das Medium PRISMA Anderen zugänglich machen. Er kann nicht nur, es muß es sogar im Interesse aller mit Technik Befassten, denn nur so kann es in der Technik einen Fortschritt für uns alle geben.

Im stillen Kämmerlein ist schon so manches Genie verkümmert...

So, nun zur ersten Frage:
Was kann man in PRISMA alles veröffentlichen ?

- **Programme**
- **Lösungsvorschläge** für Problemstellungen im Hard- und Softwarebereich; d.h. Algorithmen für mathematische oder ähnliche Probleme (Matrizen, Sortierung etc.) bzw. Schaltungen zum Selbstbauen oder mechanische Veränderungen am Rechner selbst etc.
- **Hilferufe** auf der Suche nach Informationen; z.B. "Wer hat das Buch xyz des Autors Abc, es ist im Buchhandel nicht mehr zu bekommen" oder "Wer hat einen Lösungsvorschlag für die Lösung meines Hard/Softwareproblems ?"
- **Buchbesprechungen** bzw. Kritiken, egal ob positiv oder negativ: Wer hat schon die Zeit und Gelegenheit festzustellen, ob da ein Autor einfach nur fleißig Programme kopiert hat und ein wenig malerische Worte über das Thema seines Buches verloren hat. Diese Informationen können anderen Mitgliedern viel Geld und Nerven sparen, man selbst ist ja auch dankbar für jeden solchen Hinweis.
- **Testberichte** über neue oder alte Hardwareerweiterungen:
  Auch dies wird von allen sehr gewünscht, wer weiß denn schon, was es alles auf dem Markt gibt und ob das Modul für 349,99.- DM sein Geld auch wert ist. Probeexemplare sind ja bekanntlich hier kaum zu bekommen.
- **Berichte über Betriebssystemfehler** oder Unregelmäßigkeiten bei der Programmausführung. Vielleicht ist das Phänomen jemand anderem schon aufgefallen und ein Dritter forscht dann gründlicher nach, um dann die Ursache oder eine Umgehung des Problems zu präsentieren.

Jetzt soll bloß keiner glauben, daß das schon alles war, ich will nur nicht alles vorkauen, der Phantasie der Clubmitglieder sollen hier ja schließlich keinerlei Grenzen gesetzt werden. Man muß sich nur immer vor Augen führen, was man gerne selber lesen oder erfahren würde.
Ein Artikel ist auch eine Möglichkeit, Kontakte zu anderen Interessierten aufzunehmen, oft genug schon entstand über einen Artikel eine Verbindung zwischen z.B. zwei Studenten, weil der eine seine Erfahrungen zum Besten gegeben hatte und ein Anderer genau dieses Wissen für seine Diplomarbeit gebrauchen konnte.

Jetzt stellt sich natürlich die zweite Frage:
Welche Form muß ein Artikel haben, den man an die Redaktion (Clubadresse, bitte nur das Postfach benutzen) schickt ?

Gar keine !

Wir haben keinerlei Formvorschriften, einige Wünsche für die Erleichterung unserer Feierabendtätigkeit möchte ich aber trotzdem an den Mann bzw. an die Frau bringen (ich weiß, die Frau gehört normalerweise aus Höflichkeit an die erste Stelle).

Text

- Der Artikel / Brief sollte, falls handgeschrieben, für einen normal Sterblichen lesbar sein. Es muß natürlich keine Blockschrift sein.
  Unser Auge freut sich natürlich über jeden mit irgendeiner Maschine geschriebenen Text, weil er einfach leichter lesbar ist und Fehler vermeiden hilft.

Listings

- Listings bitte unbedingt in schwarz, die blauen Listings sind nur mit großer Mühe druckreif zu machen, mehrmaliges Kopieren ist hier oft vonnöten, um den Kontrast zu verbessern. Dabei ist das Originallisting normalerweise am besten; wer die Möglichkeit hat, der sollte von seinen Listings noch eine saubere Kopie machen, da bekanntlich das Thermopapier unter vielen Einflüssen leidet, vor allem gerade an den Knickstellen.
  Im PRISMA 5-6/90 waren in dem Artikel "Zeiterfassung und Druckutilities" blaue Listings verwendet worden, beim letzten Beispielausdruck ist schon fast nichts mehr zu erkennen, manch ein Buchstabe versank in der Kontrastlosigkeit.

  Kleiner Tip für HP48-Programmlistings: Eine Prüfsumme stellt sicher, daß man sich bei der Programmeingabe nicht irgendwo vertippt hat.

  Mit welchem Drucker Listings gemacht werden, ist prinzipiell egal, solange man alle Zeichen erkennen kann oder danach die verwendeten Sonderzeichen bezogen auf den entsprechenden Rechnerzeichensatz kurz aufgezählt werden.
- Beispiele mit genauer Erläuterung helfen gerade dem Anfänger, sich in der ihm vielleicht neuen Umgebung zurecht zu finden.

Bilder

- "Ein Bild sagt mehr als tausend Worte ..."

  Ich habe versucht, in meinem Testbericht über den HP27S im PRISMA 5-6/90 mit Hilfe vieler Bilder einen deutlichen Einblick in die Arbeitsweise des Rechners zu geben, indem ich statt einer verbalen Beschreibung der Menüs einfach die Displayanzeige als Bild präsentiert habe, dies ist am anschaulichsten.
  Wer nun glaubt, daß dies nur mit modernster Computertechnologie zu machen ist, der hat hier weit gefehlt:
  Ausschneiden und Einkleben war hier die Arbeitsmethode.
  Bei Testberichten ist oft auch ein Photo interessant, um das Objekt der Begierde zu zeigen. Da sieht

man auf einen Blick, um was es geht. Photos bitte ausschließlich in schwarzweiß, Farbbilder kann man so gut wie nicht verwenden, weil die Rasterung in der Druckerei den geringen Kontrast der Farbbilder zu einer dunklen Fläche verarbeitet:
Blau und rot nebeneinander auf dem Bild geben auf einer Rasterung eine schwarze Fläche...
Zeichnungen müssen nicht in Tusche auf Transparentpapier sein, eine sauber erkennbare Handzeichnung genügt vollkommen, über mehr freuen sich natürlich alle.
Die Größe einer Zeichnung sollte nicht größer sein als auf einer Seite Platz hat, wir können aber auch mit Hilfe des Kopierers stufenlos verkleinern oder vergrößern.

Ansonsten sind der Phantasie des Autors auch hier keinerlei Grenzen gesetzt.

Aus dem Text sollte in etwa die Gliederung des Textes oder seiner Aufteilung hervorgehen, da wir auch nicht immer alle Sachverhalte nachvollziehen können. Für Anmerkungen in Bezug auf Betonungen einzelner Passagen oder Wörter, diese z.B. fett oder kursiv hervor zu heben, sind wir immer dankbar, auch der spätere Leser wird sich dann leichter tun.

Unterstrichen wird nach guter alter Satzlehre grundsätzlich nicht, dies würde den Textfluß stören. Im Text des Artikels kann man dies natürlich trotz alle dem tun, wir werden dies meist fett setzen.
Eine besondere Erleichterung und Beschleunigung ist das Schicken des Artikels auf einem Datenträger, damit wir diesen Artikel "nur noch" in dem Publisher stilistisch nach bearbeiten müssen, ohne den ganzen Text noch einmal eintippen zu müssen, was weder sehr schön noch motivierend ist und auch nur zu Fehlern führen kann.
Das gleiche wie für den Text gilt auch für Bilder, so diese auf einem computerähnlichen Gerät erstellt worden sind. Dabei sind Vektorgrafiken einfacher nach zu bearbeiten, was das Verkleinern bzw. Vergrößern angeht. Rastergrafik (Pixelgrafik) geht natürlich genauso, wobei man hier schon auf ein passendes Format achten sollte, da bei Rasterbildern ein Verkleinern bzw. Vergrößern prinzipiell machbar ist, nur daß dabei die Qualität stark leiden kann.

Wenn ich ein Rasterbild um den Faktor 2 vergrößere, dann wird eben aus jedem einzelnen Punkt eine Gruppe von vier Punkten:

Wir können folgende Textformate verarbeiten:
- WORD 4.0 (IBM-kompatibel),
- WORD 5.0 (IBM-kompatibel),
- Dateien im Ventura-Publisher Format (IBM-kompatibel),
- ASCII-Datei ohne RETURN nach jeder Zeile, nur am Absatzende. Die Diskette kann unter DOS (IBM-kompatibel) oder unter TOS (ATARI ST und TT ab TOS 1.4) formatiert worden sein.
- That's Write 1.3 - 2.0 (ATARI ST und TT)
- First Word Plus (ATARI ST und TT)
- Dokumentenformat des Publishing Partner Master bis zur Version 2.1 (ATARI ST und TT)

Wir können folgende *Bildformate* verarbeiten:
Rastergrafik
So gut wie alles, was in der Welt von IBM-kompatiblen Rechnern, von APPLE und ATARI zu finden ist. Der Converter MegaConvert von Tommy Software kann sehr viele Formate umwandeln, PCX geht natürlich ebenso wie TIFF.
Vektorgrafik
Ich kann HP-GL Daten, die ein Vektorgrafikprogramm bei der Ausgabe auf einen Plotter erzeugt, in eine Rastergrafik zurück verwandeln und diese dann beliebig nach bearbeiten. Wer also eine solche Zeichnung mit z.B. AUTOSKETCH oder GFA-Draft bzw. Techno-CAD Drafter gemacht hat, der kann diese auf Diskette plotten.
Ansonsten können wir das DXF-Format einlesen, ebenso GEM-Metafiles. Bei DXF-Files muß man aber darauf achten, daß bei der Verwendung von Symbolen nur deren Namen übergeben werden, d.h. die Grafik fehlt. Die speziellen Schriften bei z.B. AUTOCAD 10.0 werden auch nur mit der Bezeichnung der Schriftart und dem Text der Schrift übergeben, d.h. die Schrift kann in einem anderen CAD-Programm dann eine andere Höhe und Breite haben.
Dateien der CAD-Programme Techno-CAD und des Drafters sowie *.CVG, das Vektorformat von MEGAPAINT II 3.0 professional.

Nun noch kurz eine Stichwortliste für die Erstellung eines Artikels:

1. Man nehme sich "Bleistift und Papier".
2. Man schreibe kurz die Anforderungen an die Hardware (Rechner, Module etc.) des z.B. zu beschreibende Programm auf.
3. Es folgt eine kurze Beschreibung, auf was man eigentlich hinaus will: Was macht das Programm oder für wen ist es der Lösungsvorschlag für Matrizenmanipulation gut etc.
4. Man sollte nun Punkt für Punkt das Programm mit je einem Beispiel pro Möglichkeit erläutern, damit jeder die Funktionsfähigkeit desselben nachvollziehen kann. Auch decken Beispiele Fehleingaben bei der Programmeingabe auf, darum sind sie so besonders wichtig.
5. Im Text sollten, vor allem bei Beispielen, Verweise auf Bild x oder Listing x vorkommen, da wir oft die Listings oder Bilder an anderer Stelle unterbringen müssen als der Text, in dem sie erwähnt wurden.
6. Bei Buchbesprechungen oder der Verwendung von Material anderer (Routinen) muß die Quelle mit angegeben werden. Für viele ist es auch interessant, wenn sich jemand zu bestimmten Lösungen auf Grund des Studiums eines Buches inspirieren ließ.
7. Listings oder Bilder bitte immer in schwarz, Listings auf Diskette sind noch besser, da wir sie flexibler in den Artikel integrieren können. (Datenformate siehe oben)

So, jetzt kann ich allen potentiellen Autoren viel Kreativität wünschen, jeder ist dazu in der Lage, auch wenn er tiefstes bayrisch oder plattestes Norddeutsch spricht (das ist keine Rassendiskriminierung). In der Redaktion amüsieren wir uns bestimmt nicht über einen vielleicht nicht immer perfekten Satzbau, Zeichensetzungsfehler werden nach besten Wissen und Gewissen von uns korrigiert, auch wir machen schließlich Fehler.

Eine Veränderung eines eingesandten Artikels findet nur dann statt, wenn wir um des Verständnisses willen Hinweise hinzufügen, ansonsten lassen wir die Artikel so, wie sie sind. Es soll keinerlei Bevormundung geben, das ist die Grunddevise !

Im Anbetracht des heranstürmenden HP48SX sollten HP41-Besitzer keinesfalls die Hoffnung verlieren, vielen Anwendern ist er weiterhin ein treuer Kamerad und wird es weiter bleiben. Dieser Rechner stirbt nicht so schnell, also ran an die Bleistifte, auch hier gibt es bestimmt noch einige Lösungen zu berichten.

Martin Meyer (1000)
Redaktion

< Infos Infos Infos Infos Infos Infos Infos >

Dreckfuhler

Im PRISMA 5-6/90 haben sich im Artikel "Steuern über Centronics-Schnittstelle" von Peter Jochen zwei Druckfehler eingeschlichen:

1. Im Bild 2 hat das IL-Modul für den HP71 die falsche HP-Nummer. Die richtige lautet: HP82401A.
2. Im Bild 3a sind die Pinnummern 33 und 34 vertauscht. Im Bild 3c, hier handelt es sich um eine Kopie des Handbuchs, sind diese korrekt. Wer genau hinsieht, der wird auch die Logik des Zählens sofort erkennen. Ich hoffe, daß es noch nicht zu irgendwelchen Schäden gekommen ist, sorry...

In der letzten Meldung im PRISMA 5-6/90 ist so ziemlich das Schlimmste passiert, was beim Abdrucken einer Telefonnummer vorkommen konnte:
Die letzte Ziffer muß eine 3 sein, in Worten drei.
Die korrekte Nummer lautet also:
02102/499283.
Wir bitten das Versehen zu entschuldigen. Die arme Frau, der die andere Nummer gehörte, hat sich hoffentlich nicht zu sehr aus der Ruhe bringen lassen.

mm

P R I S M A Inhaltsverzeichnis

Da es immer mehr Ausgaben unserer Clubzeitschrift PRISMA gibt, haben wir uns entschlossen, das Inhaltsverzeichnis in zwei Teile aufzuteilen:
Der 1. Teil umfaßt den Inhalt der Hefte von 1982 bis 1986.
Der 2. Teil umfaßt den Inhalt der PRISMA-Ausgaben von 1987 bis 1990, d.h. der Inhalt der Hefte des vergangenen Jahres ist seit Februar verfügbar. Dies haben einige CCD-Mitglieder auch schon gemerkt.
Ab 1987 hatten wir die Erscheinungsweise auf 6 Hefte pro Jahr umgestellt, so kam der Trennstrich zustande.
Die Schutzgebühr von 3.- DM bleibt bestehen, sie gilt jetzt für jeden Teil einzeln. Sie soll wenigstens den Kopieraufwand decken.
Wer also ein Inhaltsverzeichnis haben möchte, der muß schon deutlich machen, welches er haben möchte.
Ansonsten gehen wir davon aus, daß der Wunsch nach dem aktuelleren Teil, d.h. 1987-90 selbstverständlich ist.

Redaktion

DESKJET 500

Im letzten PRISMA hatte ich ja geschrieben, daß der DESKJET 500 der alleinige Nachfolger des DESKJET und des DESKJET PLUS ist, die nicht weiter produziert werden.
Im Handbuch des DESKJET 500 ist von einem Umrüstkit die Rede, mit dessen Hilfe der "normale" DESKJET PLUS zu einem DESKJET 500 umgebaut werden kann. Dieses Kit scheint aus dem sogenannten motherboard, d.h. der Hauptplatine mitsamt dem Programm-ROM sowie dem um die Fonts CG Times und Letter Gothik erweiterten character-ROM zu bestehen.
Den abgespeckten Handbüchern des 500 ist ein Bestellformular für das Buch **"DESKJET UNLIMITED"** beigelegt.
In Erwartung der vollmundigen Beschreibung im Prospekt habe ich mir dieses besagte Buch gleich bestellt.
Zur Information: Ich besitze den DESKJET PLUS.
Nun, nach gut 4 Wochen bekam ich das heißersehnte Exemplar aus England zugeschickt (ich hatte es in den USA bestellt), meine Suche nach den versprochenen Informationen über den Aufbau der Softfonts war aber in dieser grenzenlosen Information nicht enthalten.
Kurz der Inhalt dieses angepriesenen Werkes:

1. Kurze Einleitung
2. Hardware-Beschreibung
3. Der Geist der Maschine (Grundlagen der Philosophien des Drukkers)
4. Dienstprogramme
5. Fonts (Schriftarten) und deren Anwendung
6. Fonts für den DESKJET
7. Gebrauch der Softfonts
8. Anwenderprogramme (Textverarbeitung, Tabellenkalkulationen etc.)
9. Einstellen des DESKJET von MS-DOS aus
10. Verarbeitung von Texten
11. Desktop Publishing
12. Grafik-Software
13. Chart-Software
14. Tabellenkalkulation
15. Datanbank-Software
16. Formularprogramme
17. MS-Windows
18. Spezielle Anwendungen
19. Drucken von Hardcopies (Bildschirmkopien)
20. Postscript
21. Drucken mit dem Apple-Mac
22. Programmierung des DESKJET
23. Fehlerbehandlung

A. Spezifikation
B. Zeichensätze
C. Liste der besprochenen Produkte
D. Quellen für Schriften
Stichwortliste

Wie man unschwer erkennen kann wird in diesem Buch vor allem auf die Anwendung des Druckers eingegangen, alles in Englisch, also Vorsicht.
Die Kapitel 22-B sind eine fast originalgetreue Kopie des Programmierhandbuchs des DESKJET PLUS, das bei der Lieferung des DESKJET 500 ja fehlt. Wer also eigene Druckertreiber ändern oder erstellen will, der kommt um die Anschaffung dieses Buches nicht herum, außer, er kennt jemanden mit einem PLUS.
Das Kerning wird leider so gut wie nicht beschrieben. Die Konzentration des Buches auf das Anpreisen von PC-Software merkt man recht schnell, es werden auch Preise und Bezugsquellen genannt. Diese sind natürlich für Europäer absolut uninteressant.
Einige interessante technische Details der Drucktechologie sind sehr gut beschrieben, ein schöner Exkurs in die Ideenwelt der HP-Ingenieure.
Wer heiße Neuigkeiten über den Aufbau der Softfonts erwartet hat, der wird herb enttäuscht, kein Wort darüber. Dies wurde wohl einer dritten Firma verkauft...
Alles in allem ist dieses Buch für den absoluten Anfänger mit guten Englischkenntnissen empfehlenswert, wenn er keinen Zugang zu dem Programmierhandbuch des "PLUS" hat.
Ansonsten ist es der absolute Hohn, daß man für einen solchen Werbeprospekt, etwas anderes ist dieses Buch im Kern nicht, auch noch etwa 36.- DM bezahlen muß.
Ich habe noch nie ein Buch gesehen, daß so viele Softwareprodukte rund um ein Hardwareprodukt anpreist wie ein Reklamezettel des Lebensmittelmarktes nebenan. Daß zusätzlich noch nützliche Information zu finden ist, das macht den Sachverhalt nicht viel besser.
Es findet sich eine neue Methode, den Kunden mit dem Beilegen von Informationen zu weiteren Geldopfer zu animieren. Dafür bezahlt er dann auch noch.

mm

Pas de deux Teil IV

oder: Wie man auf dem 28S/48SX die Matrixfunktionen anwendet.

von Ralf Pfeifer

Der folgende Artikel beschäftigt sich vor allem mit Matrizen und deren Bearbeitung, da die Modelle 28S/48SX zwar entsprechende Funktionen für reelle/komplexe Matrizen besitzen, aber die Handbücher die Kenntnis der mathematischen Theorie voraussetzen.

Zum einfacheren Verständnis erst mal die folgende Sprachregelung: Wenn ich im folgenden von Matrizen, Vektoren und Determinanten schreibe, beziehen sich diese Begriffe auf die Mathematik und die dort übliche Darstellung. Aber die Begriffe Matrixobjekt (kurz MOB) und Vektorobjekt (kurz VOB) bezeichnen immer Datentypen, die im Rechner herumlungern.

Über MOB und VOB ist zunächst zu sagen, daß sie zuerst für die schnelle Bearbeitung von Zahlen gedacht sind, und daher ihrerseits nur reelle und komplexe Zahlen speichern können, aber keine algebraischen Objekte. Jede Zahl braucht 8 Bytes, bei komplexen Zahlen je 8 Bytes für Real- und Imaginärteil. Auch die ganzen Zahlen von -9...9, die sonst nur 2,5 Bytes brauchen, erhalten im MOB oder VOB 8 Bytes. In der Mathematik ist eine Matrix zunächst nur eine Anordnung von Elementen, die nicht nur Zahlen, sondern z.B. Funktionen (wie Jacobi-Matrix) sein können. Ähnliche Darstellungen gibt es für Determinanten (eigentlich nur eine Funktion, die man auf eine quadratische Matrix anwendet) oder Tensoren, doch der 28/48 speichert auch diese als MOB ab.

Hier aber zunächst einige Operationen, die nicht im Handbuch stehen.

Es ist klar, daß man das ganze MOB oder VOB mit RCL, EVAL oder NUM zurückrufen und mit STO speichern kann. Aber wie geht das bei einem einzelnen Element dieser MOB/VOB?

So wie bei Funktionen, z.B. erhält der Sinus zwei Klammern, welche das Objekt seiner Begierde einschließen, beim X also 'SIN(X)'. Haben wir ein MOB (z.B. unter dem Namen MAT1 gespeichert) und wollen das Element an der dritten Zeile und der zweiten Spalte, so muß man 'MAT1(3;2)' in Ebene 1 ablegen und EVAL oder →NUM ausführen (die Zeilenzahl kommt zuerst in die Klammer, dann das Trennzeichen ";" oder "." je nach Wahl von Flag -51, dann die Spaltenzahl). RCL funktioniert hier nicht, EVAL bewirkt nichts, wenn MAT1 nicht existiert und →NUM bleibt nur dann ohne ERROR, wenn MAT1 existiert und eine Reelle/komplexe Zahl auswirft.

Bei VOB (z.B. unter dem Namen VEC1 gespeichert) geht es fast genauso, um das dritte Element zurückzurufen erwartet der Rechner 'VEC1(3)' in Ebene 1, dann EVAL oder →NUM. Hält man sich an diese Regel von 'Name(Index)', dann kann man auch mit dem Befehl STO auf MOB und VOB einwirken, oder diese indizierten Namen in algebraischen Objekten verwenden. Übrigens: Listen lassen sich genau wie VOB indizieren!!

Leider funktionieren die Befehle STO+, STO/, SNEG usw. nicht so, ein bedeutender Rückschritt gegenüber dem HP-15C von 1982. Als Indizes im Namen erlauben 28/48 auch Variable (lokal & global) oder Formeln. Außerdem bietet der 48SX noch die Funktion APPLY (im Handbuch nicht beschrieben), wenn nämlich die Indizes und der Name getrennt vorliegen, z.B. 'MAT1' (in Ebene 1) und die Indizes (in einer Liste in Ebene 2), hier { 3 2 }, kann man diese mit APPLY zu 'MAT(3;2)' verbinden. Falls möglich sollte man in Programmen jedoch auf GET und PUT zurückgreifen, weil das weniger Platz und Zeit braucht.

28S und 48SX bieten eine Reihe von mathematischen Operationen, welche sie auf MOB und VOB anwenden können:

1. Addition und Subtraktion sind erlaubt, falls beide Objekte die gleiche Dimension haben. Hier addiert/subtrahiert der Rechner jeweils zwei Elemente, die die gleiche Position haben. Das Ergebnis hat die gleichen Dimensionen wie die Eingaben.
2. Multiplikationen mit einer Zahl (reell oder komplex) führt der Rechner auch mit jedem MOB/VOB aus, wobei er jedes Element mit der Zahl multipliziert. Vorsicht: In der Mathematik ist es üblich Determinanten mit geraden Strichen zu kennzeichnen, Matrizen dagegen mit runden Klammern. Wenn man eine Determinante mit einer Konstanten multipliziert, dann darf man nur eine (beliebige) Spalte oder Zeile mit dieser Konstanten multiplizieren! Speichert man also eine Determinante in einem MOB, darf man die hier beschriebene Operation nicht einfach durchführen!
3. Die Multiplikation zweier MOB oder VOB ist eine sehr interessante Funktion, da sich Produkte und Summen hier oft ohne großen Aufwand berechnen lassen. Hier aber wieder eine Warnung: Matrizen lassen sich nur verketten (multiplizieren), wenn die Spaltenzahl der ersten Matrix gleich der Zeilenzahl der zweiten Matrix ist. Praktisch bedeutet das, daß z.B. die Matrix A (6 Zeilen 5 Spalten) und die Matrix B (5 Zeilen, 2 Spalten) miteinander verketten lassen, aber BA nicht definiert ist! Das Kommutativgesetz (Vertauschungsgesetz) gilt bei der Matrizenmultiplikation nicht! Allgemein gilt **AB** = **BA** nicht. Auch über das Ergebnis kann man Aussagen machen, es ist wieder eine Matrix, die soviele Zeilen wie A und soviele Spalten wie B hat. Diese Regel muß man beachten, wenn man zwei MOB in den Rechner eintastet. Außerdem darf man aus **AB** = **0** nicht schließen, daß **A** oder **B** Null wären!

   VOB betrachtet der Rechner als MOB mit nur einer Spalte. Bei dessen Multiplikation muß das MOB immer in Ebene 2 und das VOB immer in Ebene 1 stehen, das Ergebnis ist ein VOB. Über den Algorithmus zur Berechnung siehe späteres Beispiel.
4. Matrixinversion (INV oder 1/X) geht nur mit quadratischen MOB. Die einzige Anwendung für inverse Matrizen stellt die Auflösung von Gleichungssystemen dar, wenn man mehrere rechte Seiten hat, also z.B. das Gleichungssystem Ax = b (A = quadratische Koeffizientenmatrix, x = gesuchte Lösungen, b = rechte Seiten, Konstante) löst man so auf: $\mathbf{x} = \mathbf{A^{-1}b}$ (A^{-1} ist die inverse Matrix mit gleicher Dimension wie A, falsch wäre $\mathbf{x} = \mathbf{bA^{-1}}$ da hier die Matrixmultiplikation zwischen b und A^{-1} nicht definiert ist). Wenn sich **A** nicht ändert, kann man $\mathbf{A^{-1}}$ berechnen, speichern, und dann immer wieder mit einem neuen **b** multiplizieren und erhält dazu passend ein neues **x**. Dieser Weg ist aber wirklich nur dann empfehlenswert, wenn die verschiedenen rechten Seiten (= **b**) nicht gleichzeitig bekannt sind, sondern erst im Laufe der Rechnung herauskommen.

   Die Rechnung führt man so aus: Ebene 1 enthält das MOB mit den Koeffizienten (**A**), Ebene 2 die rechte Seite (**b**) und INV SWAP* berechnet die Lösung. Der Nachteil der inversen Matrix: In technischen Problemen fallen oft spezielle Koeffizientenmatrizen an (z.B. Bandmatrizen), die viele Nullen enthalten. Leider verschwinden diese praktischen Nullen in der inversen Matrix, weshalb man lieber eine sogenannte LR-Zerlegung durchführt. Damit wären wir auch schon bei der

5. Division von Matrizen, die in der Mathematik nicht definiert ist. Bei HP bedeutet die Division aber die Lösung von linearen Gleichungssystemen nach der LR-Methode, bei der die Koeffizientenmatrix in zwei Dreiecksmatrizen zerlegt wird. Beim HP-15C war diese Zerlegung noch zugänglich, die Modelle 28S und 48SX verbergen sie schamhaft.

   Zur Division verfährt man wie folgt: Die Koeffizientenmatrix (**A** als MOB) in Ebene 1, die rechte Seite (**b** als MOB mit einer Spalte oder als VOB) in Ebene 2 und dann /; Man kann mit dieser Funktion auch mehrere rechte Seiten (**b**) zu einer festen Koeffizientenmatrix (**A**) auf einen Schlag lösen. Dazu ein schönes Beispiel aus dem HP-15C Bedienungshandbuch. Ein gewisser Silas Farmer (s. Nr. 2) bekommt für seinen Kohl 24 Pf je kg und 86 Pf je kg Broccoli. Weiterhin erfährt man, was Mr. Farmer in drei Wochen jeweils insgesamt verkauft hat (Kohl + Broccoli) und was ihm insgesamt bezahlt wurde (wieder Kohl + Broccoli).

   Hier kann man folgendes Gleichungssystem für zunächst eine Woche aufstellen: Gleichung 1 bilanziert die "Deutschmarks", denn Kilopreis für Kohl mal x kg des verkauften Kohls plus Kilopreis des Broccoli mal y kg des Broccoli ergibt die Einkünfte in der Woche (Gleichung 1) und die Gleichung 2 bilanziert das Gesamtgewicht, denn x kg Kohl plus x kg Broccoli ergeben die verkauften "Kilogramms".

   Man sieht leicht, daß sich die linke Seite (die Koeffizienten) nicht ändert, wohl aber Gesamteinkommen und Menge (rechte Seite). Daß die Lösung einfach ist, zeigt das Beispiel, man speichert die Gesamteinnahmen der drei Wochen in die erste Zeile eines MOB, die Gesamtgewichte in dessen zweite Zeile und bringt es in Ebene 2, die Koeffizientenmatrix kommt in Ebene 1 und / liefert eine neue Matrix, in der die erste Zeile die Kohlgewichte der drei Wochen zeigt, die zweite Zeile die Broccoligewichte - Alles klar ?!

6. Um zwei Zahlen zu vergleichen, gibt es Befehle wie < oder ≠. Um Matrizen zu vergleichen, benutzt die Mathematik Normen (erkenntlich am Doppelstrich vor und hinter dem Namen der Matrix). Auch 28S und 48SX halten die üblichen Normen bereit. ABS berechnet die euklidische Norm, hier quadriert man die Beträge aller Elemente des MOB/VOB, addiert sie auf, und zieht aus der Summe die Quadratwurzel. Die Zeilensummennorm (RNRM) bildet zuerst die Beträge aller Elemente in einem MOB und addiert diese zeilenweise auf. Von diesen Werten wählt RNRM dann den Größten aus und gibt ihn aus, diese Zahl repräsentiert also nur eine Zeile eines MOB. Die Spaltensummennorm (CNRM) unterscheidet sich von RNRM nur dadurch, daß es die Beträge der Spalten statt der Zeilen aufsummiert und dann den größten Spaltenwert auswirft.

   Wegen dieser Ähnlichkeit kann man statt CNRM auch TRN RNRM und statt RNRM auch TRN CNRM programmieren. Wendet man CNRM auf ein VOB an, erhält man das Element mit dem größten Betrag, und RNRM liefert die Summe der Beträge aller Elemente in einem VOB. Zum Beispiel kann man die Normen anwenden, wenn man die Kondition einer quadratischen Matrix ermitteln will. Man berechnet NORM LASTARG INV NORM * (NORM steht hier entweder für ABS oder CNRM oder RNRM) und erhält eine Zahl. Je größer diese Konditionszahl ist, desto schlimmer schlagen Rundungsfehler bei den Ergebnissen von Operationen mit dieser Matrix zu. Eine weitere Anwendung bietet sich für die Umwandlung von komplexen MOB/VOB in reelle an. Ergibt nämlich DUP IM NORM eine Null, so sind alle Imaginärteile Null, das MOB/VOB besteht nur aus reellen Zahlen.

In PRISMA 89.4.37 habe ich das Buch "HP-28 Insights" von William C. Wickes vorgestellt. Auch für 48SX-Besitzer enthält dieses Buch eine ganze Menge an Ideen, so z.B. zur symbolischen Algebra, also dem Rechnen mit Variablen statt nur mit reellen/komplexen Zahlen. Zu den Ideen von Bill Wickes gehören auch die symbolischen Matrizen.

Wenn man sich die Darstellung des MOB im Rechner ansieht, so vertauscht Wikkes nur die eckigen Klammern der MOB mit den geschweiften der Listen (im folgenden nenne ich diese symbolischen Matrixobjekte kurz SMOB). Dem Anwender kommt also alles bekannt vor.

Die folgenden Programme habe ich speziell für den 48SX optimiert, aber sie laufen oft ohne Änderungen auch auf dem 28S. Daß die Originalprogramme von Wickes manchmal erheblich länger sind, als meine hier abgedruckten Versionen, hat zwei Gründe: Erstens hat Wickes ein Lehrbuch und kein Lösungsbuch geschrieben, die Programme lassen sich gut nachvollziehen. Und zweitens muß Wickes auch auf die Spezialitäten des 28C Rücksicht nehmen (klar, als Entwickler des 28), der aber ohnehin nicht zur Speicherung von Programmsammlungen taugt.

Im folgenden habe ich auch die Namen der Wickes-Programme übernommen, das S am Anfang steht dabei immer für "symbolische Lösungen möglich, Daten müssen in einem SMOB eingegeben werden".

Zur Berechnung von Determinanten gibt es den Entwicklungssatz von Laplace (Nr. 3). Diese Methode ist ziemlich aufwendig, denn man zerlegt dabei eine nxn Determinante in n kleinere Determinanten (wegen deren spezieller Konstruktion auch Adjunkten genannt) der Größe (n-1)x(n-1) und multipliziert diese mit einem Faktor und einem Vorzeichen. Leicht kann man sich überlegen, daß so n! Summanden mit je n Faktoren herauskommen müssen, ein Rechenaufwand, der dieses Verfahren numerisch disqualifiziert.

Doch Wickes hat dazu drei Programme entwickelt, nämlich SDET, welches SMINOR für die Adjunkte und SCOF für deren Vorzeichen benutzt. Die Programme sind durch Wechselrekursion verknüpft und deshalb sehr langsam und platzfressend.

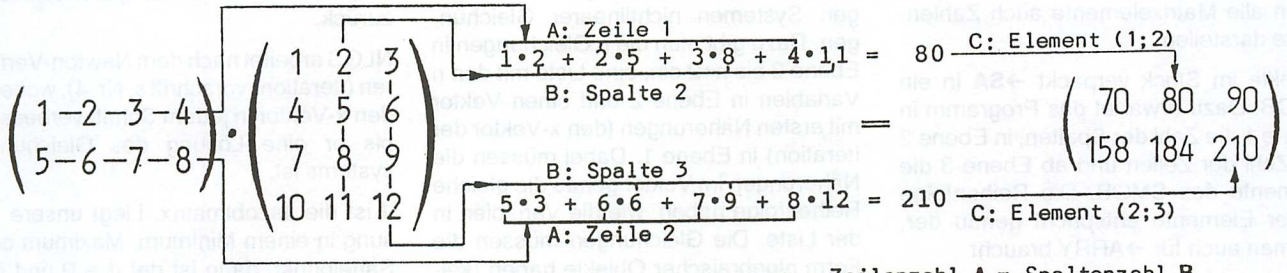
Mein Programm **SDET** braucht keine Unterprogramme und erwartet in Ebene 1 die symbolische Determinante (quadratisches SMOB). Der Geschwindigkeitsvorteil meines Programms bewirkt, daß in der Zeit, in welcher die Wickes-Version eine nxn Matrix berechnet, mein Programm die nächstgrößere schafft.

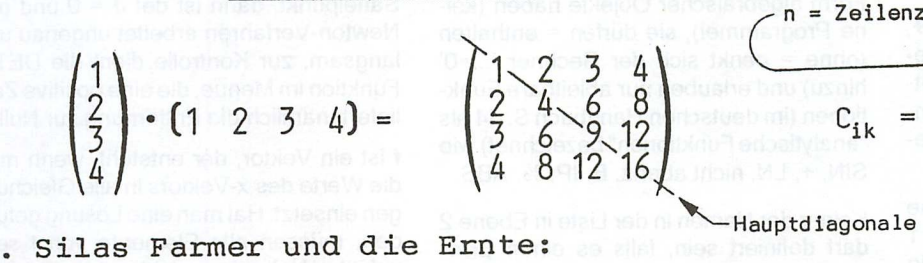
SDET besitzt eine rekursive Struktur, dennoch kann das Laplace-Verfahren nur kleine Determinanten in überlebbaren Zeiträumen berechnen, eine CRAY 1 oder CYBER 205, die 100 Millionen Multiplikationen pro Sekunde schaffen, brauchen 38 Jahre für eine 19x19 Determinante - ob da die Batterien des 48SX durchhalten?

Über die symbolischen Determinanten kann man auch die Eigenwerte einer Matrix berechnen, indem man von den Elementen der Hauptdiagonalen einfach eine Variable subtrahiert. Rechnet man dann die Determinante aus, bleibt ein Polynom übrig, dessen Nullstellen die Eigenwerte darstellen. Das sieht alles zwar sehr theoretisch aus, aber wenn Maschinen nicht vor sich hinwackeln und schwingen sollen, braucht man diese Eigenwerte. Das Programm **CEQN** erwartet in Ebene 2 das SMOB und in Ebene 1 die Variable, in der die charakteristische Gleichung (deren Nullstellen sind die Eigenwerte) erscheinen soll.

Die Multiplikation zweier SMOB bewirkt **SMUL**. Es gelten die gleichen Regeln wie bei der Multiplikation zweier MOB mit der eingebauten Funktion *, allerdings sind symbolische VOB verboten. In Ebene 2 erwartet SMUL den Faktor 1, in Ebene 1 den Faktor 2, das Ergebnis legt SMUL in Ebene 1 ab.

1. Multiplikation zweier Matrizen, A•B = C :





2. Silas Farmer und die Ernte:

Gesamteinnahmen und -gewichte von 3 Wochen:

	1. Woche	2. Woche	3. Woche	
$0{,}24\frac{DM}{kg} X_{kg\ Kohl} + 0{,}86\frac{DM}{kg} y_{kg\ Broccoli}$	120,32 DM	112,96 DM	151,36 DM	(Gl. 1)
$X_{kg\ Kohl} + y_{kg\ Broccoli}$	274 kg	233 kg	331 kg	(Gl. 2)

Matrixgleichung **A•X = B:**

$$\begin{pmatrix} 0{,}24 & 0{,}86 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_{1\ Wo.} & X_{2\ Wo.} & X_{3\ Wo.} \\ y_{1\ Wo.} & y_{2\ Wo.} & y_{3\ Wo.} \end{pmatrix} = \begin{pmatrix} 120{,}32 & 112{,}96 & 151{,}36 \\ 274 & 233 & 331 \end{pmatrix}$$

```
B A /    ergibt
[[ 186 141 215 ]
 [ 88 92 116 ]]
```

3. Entwicklungssatz für Determinanten von Laplace:

(Hier: Entwicklung nach der letzten Zeile)

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{vmatrix} = p \cdot \begin{vmatrix} a & b & c \\ e & f & g \\ i & j & k \end{vmatrix} - o \cdot \begin{vmatrix} a & b & d \\ e & f & h \\ i & j & l \end{vmatrix} + n \cdot \begin{vmatrix} a & c & d \\ e & g & h \\ i & k & l \end{vmatrix} - m \cdot \begin{vmatrix} b & c & d \\ f & g & h \\ j & k & l \end{vmatrix} ;$$

Diese Determinante als SMOB:

```
{{ a b c d }
 { e f g h }
 { i j k l }
 { m n o p }}
```

4. Nichtlineare Gleichungssysteme: Newton-Iteration.

$$X_{i+1} = X_i - J_i^{-1} \cdot f_i \quad ; \quad X = \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix}; \quad J = \begin{pmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial X_1} & \cdots & \frac{\partial f_n}{\partial X_n} \end{pmatrix}; \quad f = \begin{pmatrix} f_1(X_1, \ldots X_n) \\ \vdots \\ f_n(X_1, \ldots X_n) \end{pmatrix};$$

(Iterationsfunktion)

```
48SX NLGS
≪
  IF OVER DUP EVAL
LASTARG SIZE →LIST
≠
  THEN
"Variable löschen"
DOERR
  END 1 CF OVER { }
SWAP DUP2 SIZE → x
J v f n
  ≪ LIST→ 1
    FOR a a 1 →LIST
'x' APPLY 2 →LIST
'j' STO+ -1
    STEP { } 1 n
    START SWAP EQ→
- SWAP n 1
      FOR b OVER v
b GET ∂ COLCT j !
      IF DUP TYPE
9 SAME
      THEN ≪ EVAL
» +
      END SWAP +
-1
    STEP SWAP j !
    NEXT n DUP 2
→LIST 1 →LIST + 'j'
STO ≪ { "→NXT"
    ≪ 'x' STO 0
CONT
    » » { "→X"
    ≪ n →ARRY
    » » { "|| F ||"
    ≪ f EVAL n
→ARRY RNRM
    » » { "DET J"
    ≪ j EVAL →ARRY
DET ABS
    » » CONJ {
"EXIT"
    ≪ 1 CONT
    » » TMENU
    DO CLLCD x
DEPTH 'v' STO
      DO DUP n 1 2
→LIST RDM 1 DISP
        IFERR j
EVAL →ARRY 1 3
        START 'x'
OVER f EVAL n →ARRY
SWAP / STO-
    NEXT DROP
1 CF 0
  THEN DEPTH
v - DROPN 1
    IF 1 FS?C
    THEN
"ERROR !
Neues [ x ] wählen"
1 DISP 1 FREEZE
      ELSE SF 0
x DUP RNRM 1 / DUP
2 IFTE CON 'x' STO+
    END
  END
    UNTIL x -9
RND ROT OVER SAME
ROT KEY DROPN
LASTARG OR OR
    END IM ABS x,
DUP RE IFTE DUP 'x'
STO
  UNTIL HALT
  END 0 MENU
»
» Bytes : 865,5
Checksum : # 3296h
Ralf Pfeifer (116)
```

```
4: 'SQ(X)+SQ(Y)=25'
3: 'Y=4-SQ(X)'
2: { X Y }
1: [ 2 2 ]
   NLGS
RAD    HALT    USER
{ HOME MATH } 29.01.91  11:28:07
3:
2:
1: [ 2,84776618223
    -4,10977222865 ]
→NXT →X || F || DET J CONJ EXIT
||F|| = 0
DET J = 52,5102138421
```

Beim Startwert [4 4] konvergiert Y gegen 5,109... aber X springt. Abhilfe: Programm unterbrechen und für X komplexen Startwert, z.B. (1;1) vorgeben.

N→S (Numeric to Symbolic) verwandelt ein MOB in ein SMOB, aber **S→N** verwandelt ein SMOB nur dann in ein MOB, wenn alle Matrixelemente auch Zahlenwerte darstellen.

Objekte im Stack verpackt **→SA** in ein SMOB. Dazu erwartet das Programm in Ebene 1 die Zahl der Spalten, in Ebene 2 die Zahl der Zeilen und ab Ebene 3 die Elemente des SMOB. Die Reihenfolge dieser Elemente entspricht genau der, die man auch für →ARRY braucht.

Gerade umgekehrt funktioniert **SA→**, welches den Inhalt eines SMOB (in Ebene 1) in den Stack entleert. Ebene 1 enthält die Spaltenzahl, Ebene 2 die Zeilenzahl und ab Ebene 3 finden sich die Elemente im SMOB.

Das Programm **APLY1** wendet eine Funktion oder ein Programm in Ebene 1 auf jedes einzelne Element des SMOB in Ebene 2 an. **APLY2** erwartet in Ebene 2 und 3 zwei gleichgroße SMOBs und in Ebene 1 eine Funktion/Programm, die diese Elemente verbinden. Besteht das Programm nur aus einem - (Minus) so funktioniert APLY2 so, als hätte man zwei MOBs voneinander subtrahiert.

Wer weitere Ideen für seine symbolischen Matrixfuntionen sucht, kann sich ja bei Matthias Rabe ("Komfortable Listenverarbeitung", PRISMA 17.04.90) umsehen. Einige Programme müssen aber umgeschrieben werden, da ein SMOB ja doppelt geklammert ist.

SADD und **SSUB** addieren und subtrahieren SMOBs.

SMS multipliziert eine reelle/komplexe Zahl (in Ebene 1 oder 2) mit einem SMOB (in Ebene 2 oder 1). Im 28S muß man →STR nach ROT + in Zeile 5 einfügen.

Die eingebauten Funktionen TRN, DOT, ABS und CON wenden die Programme STRN, SCON, SABS und SDOT auf SMOBs an. **STRN** und **SABS** erwarten ein SMOB in Ebene 1, SABS ersetzt es durch die euklidische Norm, STRN durch die transponierte Matrix. **SDOT** ermittelt das Skalarprodukt der Matrizen in Ebene 1 und 2 und **SCON** erwartet in Zeile 3 die Spaltenzahl, in Ebene 2 die Zeilenzahl und den konstanten Wert in Ebene 1.

DIM berechnet die Größe eines SMOB, welches in Ebene 1 stehen sollte. Das Ergebnis ist die Zeilenzahl (Ebene 2) und die Spaltenzahl (Ebene 1).

Nach soviel Wickes habe ich noch ein paar eigene Ideen in Programme umgesetzt:

S→STR stellt SMOBs so dar, wie man es von →STR bei Arrays kennt. Dieses Programm fand beim Ausdruck der Beispiele Anwendung.

SCROSS ermittelt das Kreuzprodukt zweier symbolischer Vektoren mit genau drei Elementen.

Das Programm **NLGS** (NichtLineareGleichungsSysteme, leider nur 48SX) berechnet iterativ die Lösungen von beliebigen Systemen nichtlinearer Gleichungen. Dazu gibt man die n Gleichungen in Ebene 3 bis n+2 ein, eine Liste mit den n Variablen in Ebene 2 und einen Vektor mit ersten Näherungen (den x-Vektor der Iteration) in Ebene 1. Dabei müssen die Näherungen im Vektor genau die gleiche Reihenfolge haben, wie die Variablen in der Liste. Die Gleichungen müssen die Form algebraischer Objekte haben (keine Programme!), sie dürfen = enthalten (ohne = denkt sich der Rechner '...=0' hinzu) und erlauben nur ableitbare Funktionen (im deutschen Handbuch S. 44 als "analytische Funktionen" bezeichnet) wie SIN, +, LN, nicht aber z. B. IP, %, ABS.

Keiner der Namen in der Liste in Ebene 2 darf definiert sein, falls es einen Error gibt, muß man im VAR-Menue mit den Funktionen NXT und UP (UPDIR) nach den Namen suchen und sie löschen. Nach kurzer Bedenkzeit zeigt der Rechner die ersten sieben Variablen im Display an, so daß man den Verlauf der Iteration beobachten kann. Das Programm bricht nach Ende eines Iterationsschrittes in folgenden Fällen ab:

1. Der angezeigte Vektor (x-Vektor) hat sich in zwei aufeinanderfolgenden Iterationen nicht geändert.
2. Eine beliebige Taste außer ON wurde gedrückt. Falls die Werte nicht konvergieren, kann man so das Programm kontrolliert unterbrechen und neu starten.
3. Zwei aufeinanderfolgende Iterationen führten zu einem Error.

Nach dem Programmhalt steht in Ebene 1 der zuletzt berechnete x-Vektor (den braucht man nicht zu speichern, er verschwindet nicht) und der 48SX zeigt ein Menue mit folgenden Funktionen an:

→NXT erwartet einen neuen x-Vektor als Startvektor für eine weitere Iteration (diese Systeme können zwischen Null bis unendlich vielen Lösungen haben).

→X faßt die Zahlen in Ebene 1 bis n (n = Anzahl der Gleichungen/Variablen) zu einem Vektor zusammen.

||**F**|| setzt die Werte des ausgegebenen x-Vektors in die Gleichungen ein und sucht das Ergebnis mit dem größten Betrag heraus, dieser Wert sollte möglichst nahe bei Null sein.

DET J berechnet den Wert der Jacobideterminante. Dieser Wert sollte möglichst weit weg von Null sein.

Die eingebaute Funktion **CONJ** hilft, falls der Rechner eine komplexe Lösung anbietet. Man kann mit DUP CONJ →NXT den konjugiert komplexen x-Vektor als neue Lösung ausprobieren.

EXIT beendet das Programm. Führt man nach EXIT sofort LAST STACK aus, so erhält man immer die Eingaben für NLGS zurück.

NLGS arbeitet nach dem Newton-Verfahren (Iterationsvorschrift s. Nr. 4), wobei es den x-Vektor in jedem Schritt verbessert, bis er eine Lösung des Gleichungssystems ist.

J ist die Jacobimatrix. Liegt unsere Lösung in einem Minimum, Maximum oder Sattelpunkt, dann ist det **J** = 0 und das Newton-Verfahren arbeitet ungenau und langsam, zur Kontrolle dient die DET J Funktion im Menue, die eine positive Zahl liefert, nämlich die Entfernung zur Null.

f ist ein Vektor, der entsteht, wenn man die Werte des x-Vektors in alle Gleichungen einsetzt. Hat man eine Lösung gefunden, müssen alle Elemente von f sehr nahe bei Null sein, weicht nur ein Element besonders von Null ab, ist die ganze Lösung unbrauchbar. Um das zu prüfen verwendet man die Funktion ||F|| (die Doppelstriche bedeuten "Norm von f"), welche eine positive Zahl ermittelt, die man sich in der komplexen Ebene als Radius eines Kreises um Null vorstellen kann, in dem alle Werte von f liegen.

Um den Rechenaufwand zu verringern und das Verfahren erfolgreicher zu machen, habe ich folgenden Trick eingesetzt: NLGS berechnet pro Schritt einmal die inverse Jacobimatrix (viel Arbeit!) und verwendet sie dann für drei Iterationen.

Damit kein Error das Programm stoppt, lasse ich die Iteration zwischen einer IFERR...THEN Struktur laufen. Nach dem ersten Error sucht sich das Programm das betragsgrößte Element aus dem x-Vektor und addiert davon 1 % zu allen Elementen. Im allgemeinen reicht das aus, da der 48SX bei Funktionen wie LN, ASIN oder Potenzen mit komplexen Zahlen rechnet, so daß bei vielen Funktionen, die beim HP-41 nur in bestimmten reellen Bereichen definiert waren, im HP-48SX einzelne singuläre Punkte übrigbleiben.

Die Auswertung der Gleichungen macht also da kaum Probleme. Der Wert der Jacobideterminante darf nicht Null werden, da sonst die Inverse nicht berechnet werden kann. Falls die Gleichungen nicht unabhängig voneinander waren ist det **J** immer Null!

Bei den trigonometrischen Funktionen muß man den Winkelmodus vor dem Start von NLGS wählen, da sonst bei späterem Wechsel die falschen Ableitungen benutzt werden.

Bei manchen Gleichungssystemen kann es passieren, daß der x-Vektor nicht konvergiert, daß einige Werte des x-Vektors schnell konvergieren, andere aber wild springen (hier sollte man die springenden Werte durch neue komplexe Startwerte ersetzen), oder alle zunächst gut konver-

```
28S SDET
    « LIST→ DUP 1
002   IF SAME
      THEN GET
004   ELSE DUP 2
        IF SAME
006     THEN DROP +
    LIST→ ROLL ≠ ROT ROT
008 * -
        ELSE ROLLD LAST
010 1 - →LIST LAST → a n
          « 0 + LIST→ 2
012         FOR j a
    LIST→ 1 SWAP
              START
014 LIST→ j ROLL DROP2 n
016 →LIST LAST ROLLD
            NEXT n
018 →LIST SDET j 1 +
    ROLL j 2
020           IF MOD
              THEN NEG
022           END ≠ + -1
            STEP
024       »
        END
026   END
    »  Bytes : 233
028 Checksum : # 6EB5h

48SX SDET
    « LIST→
002   CASE DUP 1 SAME
        THEN GET
004     END DUP 2 SAME
        THEN DROP +
006 LIST→ ROLL ≠ ROT
    ROT ≠ -
008     END ROLLD
    LASTARG 1 - →LIST
010 LASTARG → a n
        « 0 + LIST→ 2
012       FOR j a LIST→
    1 SWAP
014           START LIST→
    j ROLL DROP2 n
016 →LIST LASTARG ROLLD
            NEXT n
018 →LIST SDET j 1 +
    ROLL
020           IF j 2 MOD
              THEN NEG
022           END ≠ + -1
            STEP
024       »
      END
026 »  Bytes : 223
    Checksum : # 7EE4h

1:  {{ 'SIN(X)' 1 0 }
     { 1 'SIN(X)' 1 }
     { 0 1 'SIN(X)' }}

SDET

1:  '-SIN(X)+(SIN(X)≠
    SIN(X)-1)*SIN(X)'

EXPAN COLCT

1:  'SIN(X)^3-2*SIN(X)'

48SX CEQN
    «
002   IF OVER TYPE 5 ≠
      THEN N→S
004   END + LIST→ 1 - →
    x n
006   « 1 n
        FOR j n ROLL j
008 DUP2 GET x - PUT
      NEXT n →LIST
010 SDET x n TAYLR
      »
012 »  Bytes : 129
    Checksum : # B4E3h

1:  {{ 0 2 a }
     { 1 -1 a }
     { 'SQ(a)' 0 1 }}
1:              'X'

CEQN

1: '(a≠2+a)≠SQ(a)-2+(a≠
   SQ(a)+3)*X-6/3!*X^3'

48SX DIM
    « SIZE LASTARG 1
002 GET SIZE
    »  Bytes : 22,5
004 Checksum : # 592h

1:  {{ 2 4 8 16 }
     { 3 9 27 81 }
     { 4 16 64 256 }}

DIM

2: 3  (Zeilenzahl)
1: 4  (Spaltenzahl)

48SX S→N
    « LIST→ DUP2 + ROT
002 SIZE + 2 ROT
      START +
004   NEXT EVAL 2 →LIST
      IFERR →ARRY
006   THEN →SA
      END
008 »  Bytes : 61,5
    Checksum : # 22Dh

48SX N→S
    « ARRY→ LIST→ DUP2
002 DROPN →SA
    »  Bytes : 26,5
004 Checksum : # 3199h

1:          {{ 1 2 3 }
             { 4 5 6 }}
  S→N
1:          [[ 1 2 3 ]
             [ 4 5 6 ]]
  N→S
1:          {{ 1 2 3 }
             { 4 5 6 }}

48SX →SA
    « ≠ LASTARG → n m
002   « →LIST 1 n
        START 1 m SUB
004 LASTARG + OVER SIZE
    SUB
006     NEXT DROP n
      » →LIST
008 »  Bytes : 77,5
    Checksum : # 8504h

48SX SA→
    « LIST→ DUP2 SWAP
002 SIZE 2 →LIST 1 ROT
      START +
004   NEXT LIST→ DROP
    »  Bytes : 42,5
006 Checksum : # BB5Dh

6: 'SIN(X)'
5: 2
4: -5
3: (2;2)
2: -7
1: 3

2 3 →SA

1:     {{ 'SIN(X)' 2 -5 }
        { (2;2) -7 3 }}

   SA→

6: 'SIN(X)'
7: 2 ←——→ Zeile 1
6: -5
5: (2;2)
4: -7 ←——→ Zeile 2
3: 3
2: 2 ←—— (Zeilenzahl)
1: 3 ←—— (Spaltenzahl)

48SX APLY1
    « OVER DIM → f z s
002   « LIST→ 1 SWAP
        START LIST→ 1
004   SWAP
          START f EVAL
006   s ROLL
        NEXT LASTARG
008 →LIST z ROLL
      NEXT LASTARG
010   » →LIST
    »  Bytes : 96
012 Checksum : # 220h

2:              {{ X Y }
                 { A B }}
1:                « LN »
  APLY1

1: {{ 'LN(X)' 'LN(Y)' }
    { 'LN(A)' 'LN(B)' }}

48SX APLY2
    « OVER DIM → b f z
002 s
      « LIST→ 1 SWAP
004     FOR j z ROLL
    LIST→ 1 SWAP
006       FOR k s ROLL
    b j GET k GET f
008 EVAL
        NEXT s →LIST
010   NEXT z →LIST
      »
012 »  Bytes : 132
    Checksum : # 679Eh

    Ralf Pfeifer (116)
    Rubensstr. 5
    5000 Köln 50

3:          {{ 1 2 3 }
             { 4 5 6 }}
2:          {{ 7 7 2 }
             { 2 2 X }}
1:            « - SQ »
  APLY2
1:  {{ 36 25 1 }
     { 4 9 'SQ(6-X)'}}

48SX SMUL
    « OVER DIM 3 PICK
002 SIZE SAME / OVER 1
    GET SIZE → a b m s
004   « 1 m
        FOR j 1 s
006       FOR k a j GET
    LIST→ 0 SWAP 1
008       FOR l b l
    GET k GET ROT ≠ +
010 -1
        STEP
012     NEXT s →LIST
      NEXT m →LIST
014   »
    »  Bytes : 173,5
016 Checksum : # ABFFh

2:          {{ X }
             { Y }
             { Z }}
1:       {{ X Y Z }}
  SMUL
{{ 'X≠X' 'Y≠X' 'Z≠X' }
 { 'X≠Y' 'Y≠Y' 'Z≠Y' }
 { 'X≠Z' 'Y≠Z' 'Z≠Z' }}

48SX SMS
    « DUP
002   IF TYPE 5 SAME
      THEN SWAP
004   END RCLF STD "«"
ROT + " ≠»" + STR→
006 SWAP STOF APLY1
    »  Bytes : 72,5
008 Checksum : # BD18h

2:          {{ 2 3 }
             { 5 7 }}
1:              7,2
  SMS
1:     {{ 14,4 21,6 }
        { 36 50,4 }}

48SX SDOT
    « OVER DUP 1 GET
002 SIZE → a b s
      « SIZE 0 1 ROT
004     FOR j 1 s
          FOR k a j GET
006 k GET b j GET k GET
    ≠ +
008       NEXT
        NEXT
010   »
    »  Bytes : 121,5
012 Checksum : # 89Fh

2:          {{ 1 2 X }
             { 3 5 X }}
1:          {{ X X '2≠X' }
             { 1 1 0 }}
  SDOT
1:    'X+X≠2+2≠X≠X+3+5'

48SX SADD
    «
002   « + COLCT
      » APLY2
004 »  Bytes : 36
    Checksum : # E3E4h

48SX SSUB
    «
002   « - COLCT
      » APLY2
004 »  Bytes : 36
    Checksum : # 87B2h

2:          {{ 7 7 6 }
             { 0 0 7 }}
1:          {{ 3 6 2 }
             { 8 8 6 }}
  SSUB
1:     {{ 4 1 4 }
        { -8 -8 1 }
}

48SX SABS
    « LIST→ { } 1 ROT
002   START +
      NEXT LIST→ 0 1
004 ROT
      START SWAP ABS SQ
006 +
      NEXT ∫
008 »  Bytes : 57,5
    Checksum : # AA71h

1:          {{ 1 2 3 }}
  SABS
1:     3,74165738677

48SX SCON
    « 1 →LIST
002   WHILE DUP2 SIZE >
      REPEAT DUP +
004   END 1 ROT SUB 1
    →LIST
006   WHILE DUP2 SIZE >
      REPEAT DUP +
008   END 1 ROT SUB
    »  Bytes : 85
010 Checksum : # 8D7Bh

3:          2
2:          3
1:        'X'
  SCON
1:          {{ X X X }
             { X X X }}

48SX JACOBI
    « SWAP OVER SIZE
002 OVER SIZE → f z s
      « LIST→ 1 SWAP
004     START z ROLL {
    } 1 s
006       FOR j f j GET
    3 PICK a +
008       NEXT SWAP
    DROP
010     NEXT z →LIST
      »
012 »  Bytes : 118
    Checksum : # 7F2h

2: { 'SIN(X)' 'COS(Y)' }
1:                 { X Y }
  JACOBI
1:     {{ 'COS(X)' 0 }
        { 0 '-SIN(Y)' }}

48SX STRN
    « DUP DIM → a n m
002   « 1 m
        FOR j 1 n
004       FOR i a i GET
    j GET
006       NEXT
        NEXT m n
008   » →SA
    »  Bytes : 107
010 Checksum : # A128h

1:          {{ 1 2 3 4 }
             { 5 6 7 8 }
             { 9 0 A B }}
  STRN
1:          {{ 1 5 9 }
             { 2 6 0 }
             { 3 7 A }
             { 4 8 B }}

48SX S→STR
    « "{" + LIST→ 2
002   FOR n n ROLL →STR
    + 10 CHR " " + + -1
004   STEP 1 OVER SIZE
    2 - SUB "}" +
006 »  Bytes : 95
    Checksum : # 4CC9h

1: { { 1 2 3 4 } { 5 6 7
8 } }

S→STR

1: "{{ 1 2 3 4 }
 { 5 6 7 8 }}"

48SX VDISP
    « # 83h # 40h BLANK
002 OVER SIZE LIST→
    SWAP 9 MIN
004   FOR n OVER n GET
    1 →GROB { # 1h } n
006 6 ≠ 5 - R→B + SWAP
    REPL
008   NEXT 3 FREEZE
    →LCD DROP
010 »  Bytes : 130
    Checksum : # 93A3h

1:        [ (1;2) (3;4)
            (5;6) (7;8) ]
9 √ VDISP

(.111111111111;,222222222222)
(.333333333333;,444444444444)
(.555555555556;,666666666667)
(.777777777778;,888888888889)

48SX SCROSS
    « + EVAL 5 PICK
002 OVER ≠ 5 PICK 4
    PICK ≠ - 5 ROLL 5
004 PICK ≠ ROT 7 PICK ≠
    - 2 →LIST 5 ROLL
006 ROT ≠ ROT 4 ROLL ≠
    - +
008 »  Bytes : 95
    Checksum : # 14CEh
```

gieren, dann aber an manchen Stellen endlos um den richtigen Wert kreisen. Für diese Fällle ist der Abbruch per Tastendruck gedacht, so daß man die Iteration mit einem neuen x-Vektor starten kann.

Der 28S hat keine frei programmierbare Menuesteuerung, kann ohne | (where, wobei) Variable nicht vernüftig substituieren und wertet außerden keine Listen mit EVAL aus. Einen Nachteil, der beiden Rechnern anhaftet, will ich aber noch erwähnen: Die Ableitungsfunktion ∂ kann weder nach lokalen 'Namen' noch nach indizierten 'Namen' (z.B. 'X(3)') ableiten. Bei den übrigen zugelassenen 'Namen' setzt ∂ immer Werte ein, wenn diese irgendwo im Speicher erreichbar waren, egal wie Flag -3 steht.

Die Flags -20 und -21 ignoriert der 48SX, wenn er mit VOB/MOB arbeitet. Ein Über- oder Unterlauf eines Elements führt nie zu einem "Underflow" oder "Overflow"-Error.

Die Funktion COLCT (Zeile 19) müht sich lange und oft vergeblich, aber manchmal hilft sie, komplizierte Ausdrücke zu vereinfachen (reduziert Rundungsfehler und Rechenzeit) und manchmal beseitigt sie sogar Unstetigkeitsstellen, z.B. macht x^3/x bei Null keinen Ärger mehr.

Hier noch etwas zum "knoff-hoff" von NLGS. Um f zu ermitteln, ersetzt NLGS alle Namen, die in der Liste in Ebene 2 eingegeben wurden, durch indizierte lokale Variable, nämlich x(1) bis x(n). Dann kommen die so veränderten Funktionen, jeweils gefolgt von einem EVAL, in eine Liste, und diese wiederum in die lokale Variable f. Führt das Programm nun f EVAL aus, arbeitet der 48SX die Liste wie ein Programm ab: Er findet die erste Funktion und schreibt sie in den Stack. Dann findet er ein EVAL und setzt die Werte des Vektors in der lokalen Variable x ein. So geht das Funktion um Funktion, und zum Schluß faßt man die Zahlen im Stack mit n →ARRY zusammen. Analog berechnet das Programm auch die Jacobideterminante.

Die 48SX-Modelle der A-Version haben einen Bug bei der Matrixinversion. Wer mehr als acht Gleichungen bearbeiten will, muß deshalb den Befehl INV (Zeile 49) durch IDN LASTARG / ersetzen.

Um den Überblick über die Iteration zu verbessern, kann man das Programm **VDISP** einsetzen. Es verwendet die Schriftgröße der Statuszeile im 48SX-LCD und kann so Real- und Imaginärteile nebeneinander darstellen. VDISP erwartet in Ebene 1 einen Vektor, dessen Elemente es anzeigt. Um es in NLGS einzusetzen, muß man dort die Befehle zwischen DUP (Zeile 46) und IFERR (Zeile 48) durch VDISP ersetzen.

JACOBI berechnet die Jacobimatrix (Funktionalmatrix) von Funktionen. JACOBI erwartet die Funktionen als Liste in Ebene 2 und in Ebene 1 eine Liste der unabhängigen Variablen. Das Ergebnis ist ein SMOB, und falls beide Listen die gleiche Größe haben, ist die Jacobimatrix quadratisch und SDET kann feststellen, ob die Funktionen linear abhängig sind (falls ja, liefert SDET einen Ausdruck, der immer Null ist).

Ralf Pfeifer (116)

Pas de deux Teil V

Tips & Tricks für 28S/48SX

von Ralf Pfeifer

In diesem Artikel möchte ich weniger neue Programme vorstellen, sondern vor allem etwas zum "wie" des Programmierens schreiben.

Die Basis jeder Berechnung ist die mathematische Genauigkeit. Die Mathmatik erlaubt Zahlen jede Anzahl von Stellen, und die einzelnen Ziffern können in jedem beliebigen Muster aufeinander folgen.

Ein Rechner hat dagegen nur endlich viele Stellen, beim 28S, 42S und 48SX genau 12. Zusammen mit Exponent und Vorzeichen kennen diese Rechner also 1798 200000 000001 verschiedene Zahlen. Alle Zahlen, die nicht exakt in dieses Schema passen, runden die Rechner auf eine ihnen bekannte Zahl.

Zu den unangenehmen Effekten einer Berechnung gehört die Auslöschung, also wenn zwei Zahlen, die in den ersten Stellen übereinstimmmen, voneinander abgezogen werden.

Beispiel: Die Zahl e hat in freier Wildbahn unendlich viele Stellen, der Rechner stutzt sie aber auf 12 zurecht. Zieht man von diesem Rechner-e 2,71 ab, so bleiben nur noch neun wichtige (= signifikante) Ziffern übrig. Könnte man von echten e 2,71 abziehen, wären immer noch unendlich viele Stellen übrig, also nach wie vor volle Genauigkeit.

Egal welche Berechnungen ein Programm ausführt, meist läßt sich ein Beispiel finden, welches das Programm besonders gut löst, und ein weiteres, welches zum numerischen Totalschaden führt. Andererseits gibt es nur für einfache Probleme Programme, die wie eine "black-box" alles gleich gut berechnen (wie z.B. Primzahlensucher).

Die beste Strategie ist daher, möglichst numerisch genau zu rechnen. Dazu einige Beispiele: Statt x^2-a^2 wählt man besser (x-a) (x+a), wählt man x=1 so ist die erste Gleichung für Werte zwischen 1 und 1E-6 mit zwölf Nachkommastellen ungenau.

Nimmt man das Polynom $(x-1)^3$ und füttert es SOLVR, so kommt die Nullstelle ganz genau heraus. Multipliziert man dagegen aus, und sucht mit x^3-3x^2+3x-1 (oder nach Horner mit 'X(X(X-3)+3)-1'), dann kommen nur die ersten vier Stellen richtig heraus!

Man sollte möglichst lange mit Zahlen rechnen, die der Rechner exakt darstellen kann, meist führt das auch zu weniger Rechenoperationen: Z.B. lieber dividieren als mit dem Kehrwert zu multiplizieren, lieber das Vorzeichen des Exponenten umdrehen als zuerst potenzieren und dann Kehrwert usw.; In manchen Fällen, z.B. Bruchrechnung, kann es aber zu mehr Rechenaufwand kommen, (a-3b)/9 ist länger und genauer als a/9-b/3.

Wenn möglich, sollte man für Berechnungen eingebaute Funktionen finden, die möglichst viele Schritte intern durchführen. Wer sich an die Zeiten erinnert, in denen TI noch ernsthafte Konkurrenzmodelle zu HP baute, weiß vielleicht auch, daß sich der SR-52 und TI-59 mit 12 bzw. 13 Stellen schon an einfachen Rechenaufgaben übernahmen (2^3 war nie 8, der Tangens in der Nähe von 90° in der 3. Stelle falsch), während HP-65 und HP-67 mit nur 10 Stellen bessere Lösungen auch bei größeren Problemen lieferten.

Und jetzt das Bastelset für 28S/48SX:

1 Fehlerfallen

Programme sollten ihre Eingaben überprüfen. Mein Programm OF (PRISMA 90.3.28) berechnet aus der Jahreszahl in Ebene 1 das Osterdatum. Das Programm erlaubt aber nur Jahreszahlen zwischen 1900 und 2099. Für solche numerischen Daten kann man die Vergleichsfunktionen (z.B. ==, <, ...) einsetzen. Sie liefern entweder eine 0 oder 1. Dividiert man

durch die Null, bricht das Programm mit Error ab. Dividiert man durch 1, ändert sich die geprüfte Zahl nicht. Die Tests für OF könnte man so angehen (im Programm wurde eine ähnliche Lösung gewählt):

a) Ist die Zahl in Ebene 1 größer oder gleich 1900? Falls nein, Programm abbrechen. Man testet mit DUP 1900 ≥ /;
b) Ist die Zahl in Ebene 1 außerdem noch kleiner als 210? Falls nein, Programm abbrechen. Man testet wieder DUP 2100 < /;

Gebraucht das Programm noch Flag 1, kann man die Division (/) durch CF ersetzen. Da Flag 0 nicht existiert, führt ein unbefriedigender Test zu einer 0 in Ebene 1 und das anschließende CF zum Error. Läßt der Test die Eingabe zu, löscht CF Flag 1 für den späteren Gebrauch. Das Programm läuft einwandfrei weiter.

2 Gebrauch von IFERR

Zwischen IFERR und THEN kann man ganze Programme laufen lassen. Manchmal ist es erwünscht, daß ein Programm nach einem Error nicht planlos abbricht, z.B. bei einem Iterationsprogramm, das zufällig beim Auswerten einer Funktion durch Null dividiert hat. Hier ist ein Abbruch für den Anwender unkomfortabel, vor allem, wenn sich das Programm selbst helfen könnte (z.B. Startwert verändern).

Bei einem Programm kann man aber manchmal nicht oder nur schwer feststellen, wieviele Stackebenen nach dem Error durch Datenmüll verseucht sind. Man könnte CLEAR anwenden, das hat aber den Nachteil, daß unser Programm nicht mehr als Unterprogramm taugt, weil man vorher nie genau weiß, ob CLEAR zuschlägt.

Abhilfe schafft die folgende Struktur:

```
... DEPTH → n «
    IFERR ...(Programm) ....
    THEN DEPTH n -DROPN
    ...(weitere Aktionen bei Fehler) ...
    ELSE ... (keinFehler) ....
    END »
```

Diese Struktur mißt zunächst die Stacktiefe und speichert sie in der lokalen Variablen n. Dann läuft das Programm zwischen IFERR und THEN ab. Falls ein Error auftritt, löscht die Struktur nach THEN alle durch das fehlerhaft laufende Programm zusätzlich erzeugten Stackebenen. Danach können die Maßnahmen ablaufen, die das Programm zur Korrektur braucht. Das ELSE muß nicht eingesetzt werden.

3 Eingabe von Daten

Die eingebaute Funktion PVAR erlaubt als Eingabe sowohl einen einzelnen Variablennamen, als auch eine Liste mit Namen. Mein Programm PRP (PRISMA 90.3.28) könnte dazu testen, ob sich in Ebene 1 ein Objekt vom Typ 5 (Liste) oder vom Typ 6 (globaler Name) befindet. PRP addiert aber eine leere Liste:

{ } +

Befand sich in Ebene 1 bereits eine Liste, so ändert sich nichts, befand sich dort jedoch ein 'Name', so addiert + ihn zum Inhalt der Liste. Nach diesem Schritt habe ich in Ebene 1 auf jeden Fall eine Liste und mit LIST→ erhalte ich sogar die Anzahl der Elemente in Ebene 1.

4 Endlosschleifen

Die kürzeste Endlosschleife ist

DO...UNTIL 0 END

5 Tastaturabfragen

Manche Programme müssen in Endlosschleifen laufen. Dennoch möchte der Anwender irgendwann seine Ruhe haben. Um einen kontrollierten Abbruch ohne Datenmüll im Stack durchzuführen, empfehle ich die Schleife

DO...UNTIL KEY END DROP

Drückt man keine Taste läuft das Programm ewig, denn KEY liefert jedesmal eine 0. Sobald man jedoch eine beliebige Taste außer ON bedient, liefert KEY eine 1 in Ebene 1 und einen String (28S) bzw. Zahl (48SX) die dann DROP beseitigt (Anwendungsbeispiel s. FR13 in PRISMA 90.4.30).

Um vor solchen tastengesteuerten Endlosschleifen den Key-Buffer zu löschen verwendet man:

```
WHILE KEY
REPEAT DROP
END
```

Die ersten Tastendrücke nach dem Programmstart speichert der Rechner im Key-Buffer. Die Funktion KEY holt den ältesten heraus und löscht ihn im Buffer. Das Löschen ist nur notwendig, wenn das Programm länger läuft, bis es zur tastengesteuerten Endlosschleife kommt.

Wenn das Programm gleich auf mehrere Tastendrücke wartet, z.B. um diese als Information auszuwerten, dann hilft diese Schleife:

```
1 n
START KEY
STEP
```

n ersetzt man durch die Anzahl der erwarteten Tastendrücke. Der Rechner läuft zwischen START und STEP bis die gewünschte Anzahl von Tasten betätigt wurde. KEY ergibt entweder eine Null, dann bekommt STEP nichts zu fressen und der Rechner wiederholt die Schleife, ohne den Zähler zu ändern oder KEY liefert eine 1 und eine Zahl/String (je nach 48SX/28S). Die Eins nimmt STEP um den Schleifenzähler zu erhöhen.

6 Größe eines Arrays

Wieviele Zahlen enthält das Array in Ebene 1? Handelt es sich um einen Vektor, muß man nur SIZE 1 GET (28S) bzw. SIZE EVAL (48SX) ausführen. Handelt es sich um eine Matrix müßte man SIZE LIST→ DROP * (28S) bzw. SIZE EVAL * (48SX) ausführen.

Was aber, wenn mein Programm nicht genau weiß, um was es sich handelt? Mit SIZE LIST→ DUP2 DROPN * läßt sich die Anzahl der Elemente ohne Test berechnen.

7 Umwandlung Reell/Komplex

Bei Bedarf macht der Rechner aus reellen Zahlen komplexe. Aber auch wenn der Imaginärteil Null ist, behält er die komplexe Darstellung bei. Die folgenden Schritte führen die Umwandlung bei Bedarf durch:

IM LASTARG DUP RE IFTE

Dem LASTARG entspricht LAST auf dem 28S.

Für Matrizen und Vektoren muß man eine der drei Normen (ABS oder CNRM oder RNRM) im Rechner nehmen. Die Norm eines Arrays ist nur dann Null, wenn alle Elemente Null sind. Die Befehle lauten dann:

DUP IN ABS SWAP DUP RE IFTE

ABS ist meist schneller als CNRM oder RNRM, weshalb ich es hier vorziehe.

8 Programme in lokalen Variablen

Zunächst möchte ich mein Programm PRIM vorstellen. Es erwartet in Ebene 1 eine positive Ganzzahl, die es in Primfaktoren zerlegt. Die gefundenen Faktoren zeigt das Programm zwischendurch an, und zum Schluß gibt es die Faktorisierung als algebraisches Objekt aus. Der 28S braucht in Zeile 10 nach ROT noch ein →STR. Die Eingabe des Programms kann man sich erheblich erleichtern, wenn man dem CUSTOM bzw. CST-Menue den String (!)

" DUP2 MOD NOT d IFT "

zuweist. den kann man nämlich 9 mal gebrauchen, und der Rechner fügt ihn ohne " ins Programm ein.

PRIM braucht ein Unterprogramm, welches die geprüfte Zahl und den Teiler auswertet, falls deren Division aufgeht. Mein erster Versuch (PRISMA 88.5.44) benutzte dazu ein externes Unterprogramm 'DV', welches aber nur PRIM gebrauchte. Im alten HP-41 würde man das Unterprogramm einfach an das Hauptprogramm anhängen, aber in RPL fehlt der GOSUB und der GOTO-Befehl, um es anzuspringen.

In der hier vorgestellten Version habe ich daher das Unterprogramm an den Beginn (Zeilen 2-12) des Programms PRIM gesetzt, und zwar als eigenes Programm im Programm. Zeile 13 speichert dieses Unterprogramm dann in die lokale Variable d. Jedesmal, wenn ich d brauche, kann ich es im "Hauptprogramm" (Zeilen 14-32) aufrufen. Doch d legt das Programm nur in Ebene 1 ab, weil d eine lokale Variable ist. Eine globale Variable d würde das Unterprogramm sofort starten, bei Programmen in lokalen Variablen muß man mit EVAL nachhelfen. In PRIM macht das ein IFT-Test, der prüft (und verbraucht) nämlich den Inhalt von Ebene 2, und falls dieser Null ist, wirkt er auf Ebene 1 wie ein DROP, falls er in Ebene 2 eine andere reelle Zahl findet, wie ein EVAL. Noch eine Warnung: Wenn ein lokales Programm andere lokale Variablen aufruft, müssen diese im Listing schon vor dem Programmbegrenzungszeichen («) definiert sein!

9 Parameter für START und STEP

Im Gegensatz zu FOR-Schleifen stellt die START-Schleife dem Programm ihren Zähler nicht zur Verfügung. Es wäre egal, ob man START von 1 bis 15 oder von -5 bis 9 laufen läßt, wenn nicht die 15 ganze 10,5 Bytes, aber 1, -5 und 9 nur ganze 2,5 Bytes brauchen würden.

Wenn man es vermeiden kann, sollte man auch die Kombination START... STEP wegen des STEP-Parameters vermeiden. Der Grund ist eher allgemeiner Natur: Man sollte aus einer Schleife, egal ob FOR, START, DO usw. soviele Befehle als möglich herausziehen. Diese wiederholen sich dadurch nicht so oft, und das Programm läuft schneller. Der STEP-Befehl nimmt sich die Schrittweite bei jedem Aufruf aus Ebene 1. Dieser Wert muß keine Konstante sein. Man kann die Werte z.B. auch über Tests zur Verfügung stellen, ein MAXR-Befehl bewirkt den Abbruch, die Null bewirkt eine "Ehrenrunde" der Schleife, wobei FOR seinen Zähler nicht erhöht.

10 Zeitoptimierung

Bei alten 41er Programmen gehört es zum guten Ton, Zahleneingaben mit sowenig Ziffern wie möglich zu machen. Das sparte Platz und Zeit. Zahlen verdoppelte man mit ST+ X oder ENTER +, keinesfalls aber mit 2 *; Gerade hier hat sich der 28S/48SX verändert: Zahlen normalisieren die Rechner schon bei der Eingabe des Programms, und nicht erst, wenn dieses abläuft.

Eine mathematische Methode, die schnelle Fourier-Transformation (FFT) ermöglicht dem 28S/48SX Multiplikationen fast so schnell wie Additionen durchzuführen, so daß zwischen 2 * und DUP + zeitlich kein Unterschied mehr besteht. Empfindlich reagieren die neuen Rechner aber auf Stackmanipulationen. So ist ROT ROT fast dreimal schneller als 3 ROLLD und ROT OVER SWAP ist mehr als doppelt so schnell wie DUP 4 ROLL. Auch beim Löschen braucht DROP2 DROP2 nur ein Drittel der Zeit von 4 DROPN, ähnliches gilt für DUP und DUPN. Die zusätzlich verbrauchte Zeit ist allerdings sehr gering. Man sollte sich darüber erst Gedanken machen, wenn diese Funktionen in Schleifen oder häufig benutzten Unterprogrammen (z.B. in EQ) stehen. Hier sollte man möglichst mit Stackfunktionen arbeiten, die keine Zahlen vom Stack holen.

Potenzieren mit 2 (2 ^) braucht fast siebenfach länger als die Funktion SQ.

Programme sollten die Möglichkeit nutzen, nur im Reellen zu arbeiten, so braucht die Multiplikation von (2;0) mit (3;0) fast zehnmal mehr Zeit, wie ein einfaches 2 mal 3.

Wenn man mit Listen oder Arrays arbeitet, ist es ebenfalls oft günstiger, diese aufzulösen und in den Stack zu entleeren, als die Elemente mit GET und PUT zu holen. Dies gilt vor allem dann, wenn man alle Elemente aus Listen/Arrays bearbeitet.

11 Zerlegen von Zahlen

Der 48SX benutzt spezielle Zahlen zur Darstellung von Datum und Uhrzeit. Enthält Ebene 1 eine Zahl im HH,MMSSssss-Format, dann zerlegt man diese so:

```
IP 1 LASTARG FP %T
IP 1 LASRARG FP %T
```

(25 Bytes); Die Stunden findet man in Ebene 3, die Minuten in Ebene 2 und die Sekunden in Ebene 1.

Ähnlich trennt man das Datum (in Ebene 1) auf:

```
IP 1 LASTARG FP %T
IP LASTARG 10000 *
```

(30,5 Bytes); Je nach Wahl von Flag -42 (MM/TT/JJJJ oder TT.MM.JJJJ Format) erhält man in Ebene 3 Monat/Tag, in Ebene 2 Tag/Monat und das Jahr in Ebene 1.

Oft nützlich und praktisch ist in diesen Fällen der Einsatz der drei %-Funktionen (y befindet sich immer in Ebene 2, x in Ebene 1): %T berechnet 100*x/y, % berechnet x*y/100 und %CH berechnet 100(x/y-1). Hier ein paar Anwendungsbeispiele:

Statt..	Besser..
100 /	1 %
50 /	2 %
20 /	5 %
12,5 /	8 %

Byteverbrauch: Links je 13 und rechts je 5 Bytes.

Auch mit %T kann man sparen:

Statt...	Besser..
100 *	1 SWAP %T
50 *	2 SWAP %T
20 *	5 SWAP %T
12,5 *	8 SWAP %T

Byteverbrauch: Links wieder 13 Bytes, rechts 7,5 und wenn man die Zahl im Programm richtig plaziert, kann man sich sogar das SWAP sparen.

Eine andere Möglichkeit an die Ziffern einer Zahl zu kommen, bietet die MOD Funktion. wie komme ich an die 4 in 12345,6789 ? Na so bitte:

100 MOD sägt alles vor der 4 ab und mit 10 / IP erhält man 4. Auf dem 48SX ergibt -1 TRNC statt 10 / IP als Ergebnis 40.

12 IFT und IFTE

Diese Funktionen stellen die komprimierte IF..THEN-Struktur dar. Einsetzen sollte man sie immer dann, wenn zwischen zwei gleichen Objekten entschieden wird. Die Testbedingung (bei IFT in Ebene 2, IFTE in Ebene 3) muß entweder eine 0 (für Nein) oder eine beliebige reelle Zahl (für Ja) sein. Auf das erwählte Objekt wirken IFT/IFTE wie EVAL, auf das verstoßene Objekt wie DROP - LASTARG speichert in jedem Falle alle Argumente.

So eingesetzt habe ich IFT bereits in PRIM und IFTE bei der Umwandlung reeller in komplexe Zahlen. Hier noch ein Beispiel: Die Funktion XPON führt bei Null zu einem Error. Logisch gesehen hat XPON recht, der Null einen Zehnerexponenten zu verweigern, aber intern speichern alle HP-Modelle die Zahl Null zusammen mit dem Exponenten Null ab.

Man vermeidet den Error mit

```
DUP 1 IFTE XPON
```

Nach DUP 1 enthält Ebene 3 die eingegebene Zahl als Testbedingung. Wir erinnern uns, daß man als solche entweder eine Null (zufälligerweise unser Spezialfall) oder eine andere Zahl braucht. Wenn Ebene 3 keine Null enthält, überlebt der Inhalt von Ebene 2. Mit DUP hatten wir ja dafür gesorgt, daß sich auch hier die Eingabe wiederfindet. Die 1 ist der Ersatz für unsere Null, denn 1 XPON ergibt Null.

Noch ein Beispiel: Ebene 1 und 2 enthalten zwei Objekte, und das kleinere soll gelöscht werden. Bei reellen Zahlen geht das ganz einfach mit MAX. Zum Vergleich von Binärwerten und Strings braucht man aber die >-Funktion:

```
> LASTARG IFTE
```

Entsprechend kann man auch < einsetzen, um den "kleineren" Wert zu finden.

Das Programm **MAXI** gebraucht diese Methode, um eine Liste oder ein Array durch das größte Element zu ersetzen, wenn dieses (oder der Name, unter dem es gespeichert ist) in Ebene 1 steht. Enthält Ebene 1 eine positive Ganzzahl n, ermittelt MAXI das größte Element in den Stackebenen 2 bis n+1; Als Elemente

sind reelle Zahlen, Binärwerte und Strings zugelassen.

13 MIN und MAX

Mit diesen Funktionen kann man praktische Eingabeüberprüfungen machen. Beispiel: Die 3 Elemente einer Liste sollen im LCD angezeigt werden. Man könnte folgendes Programm verwenden (Liste in Ebene 1):

```
1 OVER SIZE
  FOR j DUP j GET j DISP
  NEXT
```

Das LCD im 28S hat 4 Zeilen, im 48SX 7 Zeilen. Hätte also unsere Liste 12 Elemente, würde DISP einen Error erzeugen. Daher folgende Änderung (beim 28S ist die 7 durch 4 zu ersetzen):

```
1 OVER SIZE 7 MIN
  FOR... s.o.
```

Die Befehle 7 MIN stellen sicher, daß die Schleife nur soviel Arbeit wie nötig macht.

Das folgende Programm implementiert den UPDIR-Befehl (Aufstieg in das nächsthöhere Directory) des 48SX auf dem 28S (keine Eingaben erforderlich):

```
«PATH DUP SIZE 1 -
 1 MAX GET EVAL »
```

Das Programm nimmt den Verzeichnispfad (eine Liste) und bestimmt deren Größe. Das vorletzte Element in der Liste ist das nächsthöhere Directory. Was aber, wenn wir bereits im HOME-Directory sind? Dann ergibt SIZE 1 - eine Null und GET führt zum Error. Mit 1 MAX erhält man in diesem Sonderfall eine 1.

14 Koordinatenumwandlung - 48SX

Im MTH VECTR Menue befinden sich einige Fuktionen zur Koordinatenumwandlung. Man kann hier wählen, ob der Rechner Vektoren mit zwei Elementen in kartesischen oder Polarkoordinaten, und Vektoren mit drei Elementen in kartesischen, Zylinder- oder Kugelkoordinaten darstellen soll. Es handelt sich hier wirklich nur um die Darstellung, denn intern speichert und verwendet der 48SX immer nur die kartesischen Koordinaten. In folgendem Beispiel wählen wir zunächst die kartesischen Koordinaten:

```
MTH VECTR XYZ
```

dann geben wir den ersten Vektor ein:

```
77 88 99 →V3
```

dann den leicht veränderten Vektor

```
77 88 √ SQ 99 →V3
```

Ganz offensichtlich sind diese Vektoren nicht gleich, SAME oder == ergeben eine 0; Jetzt schalten wir in Zylinderkoordinaten um:

```
MTH VECTR R⇶Z
```

Mit SWAP zeigt, daß beide Vektoren exakt gleich sind, dennoch ergeben SAME und == wieder eine 0. Auch in Kugelkoordinaten zeigt der 48SX zwei gleiche Vektoren an, wieder zeigt SAME und == eine 0; Das beweist, daß der Rechner intern nicht von seiner kartesischen Darstellung abwich. Wie wandelt man nun Koordinaten ineinander um? Hat man Zylinderkoordinaten gewählt (R⇶Z), und befinden sich die drei Koordinaten in den Ebenen 1,2 und 3, dann verwandelt

```
→V3 OBJ→ DROP
```

diese in kartesische Koordinaten, und

```
3 →ARRY V→
```

verwandelt kartesische in Zylinderkoordinaten.

Wählt man statt der Zylinderkoordinaten (R⇶Z) die Kugelkoordinaten (⇶⇶), dann bewirken die angegebenen Tastenfolgen die Umwandlung zwischen kartesischen und Kugelkoordinaten. Statt der Befehle →V3 und V→ kann man auch 3D verwenden, statt OBJ→ auch ARRY→. Die Umwandlung zwischen Zylinder- und Kugelkoordinaten geht so: Koordinatenmodus (R⇶Z oder R⇶⇶) wählen, dann →V3, in den anderen Koordinatenmodus wechseln (R⇶⇶ oder R⇶Z) und dann V→.

15 Vergleiche von Objekten

Besonders bei der Eingabe überprüft man, ob das eingegebene Objekt vom richtigen Typ ist.

Beispiel: Das Programm soll prüfen, ob das Objekt in Ebene 1 vom Typ 0 (= reelle Zahl), 1 (= komplexe Zahl) oder 9 (= algebraisches Objekt) ist. Alle anderen Objekte sind verboten. Man kann nun einzeln testen:

```
DUP TYPE DUP 0 SAME
OVER 1 SAME OR
SWAP 9 SAME OR
```

Nach 32,5 Bytes erhält man jetzt eine 0 oder 1 als Testergebnis. Mein Vorschlag:

```
{ 0 1 9 } OVER TYPE POS
```

braucht nur 20 Bytes und läßt sich sparsam erweitern. Dieser Struktur liegt folgender Gedanke zugrunde: Zunächst kommt eine Liste auf den Stack, die alle zugelassenen Elemente enthält. Dann kopiert man sich das zu testende Objekt (hier mit OVER) und führt POS aus. Diese Funktion vergleicht, ob sich das Objekt in Ebene 1 in der Liste der zugelassenen Objekte befindet. Falls nein, gibt POS eine Null aus (alle -Tests nehmen dies als NEIN), falls jedoch das geprüfte Element auch in der Liste steht, gibt POS die Nummer des Objekts in der Liste aus. Dies ist irgendeine ganze Zahl an 1 aufwärts, und alle Tests (z.B. THEN) akzeptieren das als JA.

16 ON...GOTO

Diese Struktur kennt man aus dem BASIC. Das was zwischen ON und GOTO steht ergibt eine positive ganze Zahl, und hinter GOTO stehen eine ganze Reihe von Zeilennummern. Ergibt sich zwischen ON und GOTO eine 1 nimmt der BASIC-Computer die erste Zeilennummer und springt dorthin. Steht zwischen ON und GOTO eine 3, nimmt der Computer die dritte Zeilennummer nach GOTO und springt dorthin. Ähnliches geht auch mit den Listen im 28S (und natürlich im 48SX), dazu folgendes Beispiel (etwas vereinfacht):

Das Programm **TYP?** überprüft mit TYPE das eingegebene Objekt und benennt es, z.B. ergibt -33 TYPE eine Null, und das Objekt heißt "Reelle Zahl"; TYP? legt eine Liste auf den Stack (Zeilen 1-7), also das, was bei ON...Goto nach GOTO kommt. Für die Objekttypen 0-2 hält diese Liste entsprechende Strings bereit, aber Arrays sollen in Vektoren und Matrizen unterschieden werden, weshalb die Liste für den Typ 3 keinen String, sondern ein Untersuchungsprogramm enthält. Ab Zeile 8 beginnt das, was in BASIC zwischen ON und GOTO steht, nämlich die Auswahl des richtigen Listenelements.

Wer glaubt, daß diese Methode nur dem 28S etwas bringt, weil der 48SX die CASE-Struktur hat, dem sei zu folgendem Beispiel geraten:

Die KEY-Funktion des 48SX gibt nur Zahlen zurück, für ein spezielles Programm braucht man eine Routine, die diese in Buchstaben verwandelt. Das Beispiel verwandelt nur einige ausgewählte Tastencodes, nämlich A, B, X, Y, 0 und 1.

In Zeile 1 des Programms **KEY1** wartet 0 WAIT auf einen Tastendruck, den es mit einem Tastencode identifiziert. Dann legt KEY1 eine Liste mit den zugelassenen Tastencodes auf den Stack und sieht nach, an welcher Listenposition der von WAIT gelieferte Tastencode (mit IP vom Nachkommateil befreit, sonst gibts hier Probleme) steht. In Zeile 3 kommt die zweite Liste auf den Stack. Sie enthält die Buchstaben in der gleichen Reihenfolge wie die in der ersten Liste passenden Tastencodes. Ein SWAP GET holt dann den richtigen Buchstaben.

Verbessert man KEY1, stellt man fest, daß man Buchstaben am einfachsten in Strings aufbewahrt. Die zweite Liste weicht daher dem String "ABXY01" (gleiche Reihenfolge!) und DUP SUB ersetzt GET. Eine undefinierte Taste führt bei KEY1 zum Error (wegen 0 GET), KEY2 gibt das erste Zeichen im String als Lösung aus.

17 Begrenzte Rekursion

In Pas de deux 2 (PRISMA 4/90) habe ich Beispiele aus der Mathematik zur Rekur-

```
48SX PRIM
     « IP ABS "" SWAP
002    « 0 ROT ROT
         DO SWAP OVER /
004  SWAP ROT 1 + ROT
     ROT
006      UNTIL DUP2 MOD
         END 4 ROLL DUP
008  SIZE "*" "" IFTE +
     OVER + 4 ROLL DUP 1
010  > "^" ROT + "" IFTE
     + DUP 3 DISP ROT
012  ROT
       » → d
014    « 2 DUP2 ≥ / DUP2
     MOD NOT d IFT DROP
016  3 DUP2 MOD NOT d
     IFT DROP 5 DUP2 MOD
018  NOT d IFT SIGN
         DO 6 + DUP2 MOD
020  NOT d IFT 4 + DUP2
     MOD NOT d IFT 2 +
022  DUP2 MOD NOT d IFT
     4 + DUP2 MOD NOT d
024  IFT 2 + DUP2 MOD
     NOT d IFT 4 + DUP2
026  MOD NOT d IFT 6 +
     DUP2 MOD NOT d IFT
028  2 + DUP2 MOD NOT d
     IFT
030      UNTIL DUP2 SQ <
         END DROP DUP
032  DUP 1 ≠ d IFT DROP2
       » STR→
034  »  Bytes : 429
     Checksum : # 8A1Dh

12 FACT  PRIM

1: '2^10*3^5*5^2*7*11'
```

Ralf Pfeifer (116)

```
28S MAXI
     «
002    IFERR RCL
       THEN DUP
004      IF TYPE
         THEN
006        IFERR LIST→
           THEN ARRY→
008  LIST→ DUP2 DROPN ≠
           END
010      END 2 SWAP
         START > LAST
012  IFTE
         NEXT
014    ELSE LAST SWAP
     MAXI SWAP STO
016    END
     »  Bytes : 112,5
018  Checksum : # ACEEh

[ 1 2 5 0 6 -2 ]   MAXI

1:  6
```

```
48SX TYP?
     « { "Reelle Zahl"
002  "Komplexe Zahl"
     "Zeichenkette"
004    « DUP SIZE SIZE 1
     SAME "r Vektor"
006  " Matrix" IFTE
     "Reelle" SWAP +
008    » } OVER TYPE 1 +
     GET EVAL SWAP DROP
010  »  Bytes : 152
     Checksum : # 5296h

-33   TYP?

1: "Reelle Zahl"
```

Rubensstr. 5, Köln 50

```
48SX KEY1
     « 0 WAIT { 11 12 46
002  52 92 82 } SWAP IP
     POS { "A" "B" "X"
004  "Y" "0" "1" } SWAP
     GET
006  »  Bytes : 136,5
     Checksum : # 1BA3h

48SX KEY2
     « 0 WAIT "ABXY01" {
002  11 12 46 52 92 82 }
     ROT IP POS DUP SUB
004  »  Bytes : 106,5
     Checksum : # 11B5h

48SX REV
     «
002    IF DUP TYPE
       THEN
004      IFERR RCL
         THEN SIZE
006  LASTARG
           IFERR ARRY→
008        THEN + LIST→
     REV →LIST
010        ELSE LIST→
     DUP2 DROPN * 1 +
012  REV →ARRY
           END
014      ELSE LASTARG
     SWAP REV SWAP STO
016    END
       ELSE 2 SWAP
018      FOR n n ROLL
         NEXT
020    END
     »  Bytes : 161
022  Checksum : # 2DEh

[ 1 2 3 4 5 ]   REV

1:  [ 5 4 3 2 1 ]
```

sion vorgestellt. Die Rekursionstiefe (die Anzahl der Selbstaufrufe) war dabei unbegrenzt. In dieser Programmsammlung befand sich aber auch das Programm SORT. Genau wie das hier vorgestellte REV ruft es sich höchstens zweimal selbst auf. REV kehrt die Reihenfolge der Elemente einer Liste oder Arrays (jeweils in Ebene 1) um, falls in Ebene 1 eine positive Ganzzahl n steht, dreht REV die Ebenen 2 bis n+1 um, und ein Name in Ebene 1 läßt REV prüfen ob darin eine Liste oder Array auf das Umdrehen wartet.

Nach vielen Experimenten habe ich als kürzeste Lösung das gefunden: REV kann nur die Reihenfolge im Stack umkehren (Zeilen 17-19). Alle anderen Fälle führt REV auf das Stackumdrehen zurück. Ein Array oder eine Liste löst REV auf, speichert deren Größe ab und schreibt in Ebene 1 die Anzahl der Elemente (Zeilen 7-13). Da so wieder der Fall "Stack undrehen" entstand, ruft sich REV selbst auf. Falls Ebene 1 einen 'Namen' enthält, holt RCL dessen Inhalt (kann ja nur Liste oder Array sein) und ruft sich selbst auf, stellt fest, daß es eine Liste/Array war, und ruft sich nocheinmal selbst auf. Ist doch ganz einfach, oder?

18 AND, OR, XOR, NOT und Tests

Diese vier Funktionen stellen an ihre Argumente die gleichen Anforderungen wie Tests: Eine Null betrachten sie als NEIN und jede andere reelle Zahl als JA. Für Tests kann man das so ausnutzen:

Statt	A 0 ==
geht auch	A NOT
statt	A 0 == B 0 ≠ AND
geht auch	A NOT B AND

Letztes Beispiel:

Einfacher als	A 0 ≠ B 0 ≠ OR
ist	A B OR.

Natürlich müssen A und B reelle Zahlen liefern, sonst kann man diese Vereinfachungen nicht anwenden.

Nebenbei verwende ich auch die Funktion ARG als Test. Für positive reelle Zahlen einschließlich Null liefert ARG immer 0, für negative reelle (und alle komplexen Zahlen) eine reelle Zahl.

19 Voreinstellung von Variablen

Beispiel: Ein Programm benutzt die globalen Variablen A, B, C, und D, die vor Programmstart auf 0 gesetzt werden sollen. Die Standardlösung:

0 'A' STO 0 'B' STO 0 'C' STO 0 'D'STO

Von den 58 Bytes braucht jeder Name 9,5 Bytes, davon gehen alleine 5 auf das Konto der Striche ('). In einer Liste entfallen diese Striche. Kürzer geht es so:

{ 0 A 0 B 0 C 0 D }

Das spart glatt 10 Bytes. In diesem Beispiel erhalten alle Variablen den gleichen Wert, mit einer START...NEXT Schleife kann man deshalb noch günstiger arbeiten:

```
{ ABCD 2 } LIST→
  START 0 SWAP STO
  NEXT
```

Das macht nur noch 40,5 Bytes.Der besondere Kniff ist die 2 am Ende der Liste. Nach LIST→ befindet sich die 2 in Ebene 2 und die Größe der Liste in Ebene 1, und zusammen geben die beiden tolle Anfangs- und Endwerte für START ab.

20 Zahlen mit Vornullen

Zur Numerierung von Ausgaben benötigt man manchmal Zahlen mit einer festen Stellenzahl.

Nehmen wir z.B. die Zeilennumerierung der Programmlistings dieses Artikels, die Vornullen besitzen. Das Programm PRP (in PRISMA 3/90) klemmt dazu die Nullen der Zeilenzahl ein. Es braucht dreistellige Zahlen, also addiert es zur Zeilennummer 100, da hier drei Nullen für unsere Zahlen frei sind. Beispiel: Die Zeilennummer 3 (bereits in Ebene 1) soll erzeugt werden:

1000 + →STR 2 4 SUB

Das Ergebnis ist der String "003" (STD-Format vorausgesetzt).

21 Rettung für Variable - 48SX

Je häufiger man den 48SX programmiert, um so öfter passiert es. Befindet man sich im VARS-Menue, speichert die orange

Vortaste und eine Menuetaste den Inhalt von Ebene 1 in die Variable. Hat man sich hier geirrt, dann hilft weder LAST STACK noch LASTCMD. Die Rettung bringt eine sinnvolle Ausnahme der LASTARG-Funktion. Bei STO speichert sie nämlich ausnahmsweise nicht den letzten Stackinhalt, sondern den letzten Inhalt der Variablen. Die Rettungsprozedur sieht also so aus:

LASTARG STO LASTARG DROP

und alles ist wie vorher. ein schöner Effekt, besonders wenn man einige hundert Bytes bereits verloren glaubte!

22 Großdruck - 48SX

Schickt man dem Infrarotdrucker die Anweisung doppelte Buchstabenbreite z.B. mit

27 CHR 253 CHR + PR1

und schaltet dann den Drucker aus, vergißt dieser die doppelte Breite, nicht aber der 48SX. Er übermittelt weiterhin nur 12 Zeichen pro Zeile. Ein besonders unangenehmer Effekt, den nicht einmal ein Systemhalt (ON und C zugleich drücken) behebt, es hilft nur

27 CHR 252 CHR + PR1

Der 28S speichert die Druckbreite nicht, da der Infrarotdrucker den Zeilenumbruch ohnehin automatisch durchführt. Offenbar ist der 48SX schon für eine neue IR-Druckergeneration vorbereitet!?

23 Buchbesprechung - 48SX

Das Buch "HP 48 Programmers Reference Manual" gibt's in Englisch für ca. 40 DM bei Händlern oder HP-DIREKT (Tel. 0130/3322, Best.-Nr. 00048-90054). Es enthält alle Befehle des HP 48, wobei zu jedem Befehl folgendes abgedruckt ist:

a Ein Stackdiagramm wie im 2. Teil des 28S-Handbuchs, mit möglichen Ein- und Ausgaben.
b Einige Sätze zur Anwendung der Funktion.
c Wechselwirkung mit Flags.
d Bemerkungen, die das Stackdiagramm und das Verhalten bei unerwarteten Eingaben erläutern.
e Etwa die Hälfte der Beschreibungen wartet außerdem mit Beispielen auf.
f Namen verwandter Funktionen.

Im Anhang findet man nocheinmal die Liste der Fehlermeldungen, Einheiten, Bedeutung der Systemflags, sowie Name und Bedeutung von Systemvariablen (z.B. PRAR).

Was mir fehlt: Ob und was LASTARG speichert, Funktionen auf der Tastatur (z.B. 2D), interaktive Menues, Objekttypen, deren Eigenarten und Byteverbrauch, Speicherorganisation des RAM/ROM, Informationen über die Hardware des seriellen und IR-Ausgangs (wie kann man damit Signale vom 28S empfangen?) und die Spiralheftung für die 504 DIN-A5-Seiten.

Dagegen erklärt das Buch den 48SX-Befehl WSLOG (Warmstart Log), der einige Strings mit Datum, Uhrzeit und Grund des letzten Warmstarts (z.B. wackelige Batterien, faule Daten im RAM) holt, die Formeln von Statistikfunktionen wie UTPN und die Datentypen 20-31.

Als HP noch den 25 oder 19C verkaufte, gab es zum Rechner ein großes Handbuch und eine gleichgroße Programmsammlung.

Die Rechner von HP wurden immer leistungsfähiger, aber die Handbücher wuchsen kaum mit. Und heute muß man ein Buch, welches eigentlich in deutscher Übersetzung jedem Rechner beiliegen sollte, noch bestellen und extra bezahlen. Schöner Mist.

Ralf Pfeifer (116)

Programmieren mit Turbo C++

Gerd Kebschull

Reihe Praxis Wissen, 1990, 430 Seiten, eine 5.25" Diskette, DM 58.-, ISBN 3-8023-0365-2, Vogel Buchverlag Würzburg

Das Buch von Gerd Kebschull richtet sich an C-Neulinge, die lernen wollen mit C zu programmieren. Für Anwender, die die Sprache C bereits beherrschen und die neuen Möglichkeiten von Borlands Turbo C++ Compiler kennenlernen wollen ist es weder gedacht noch geeignet.

C ist eine Sprache für den fortgeschrittenen Programmierer, der seine ersten Gehversuche in anderen Hochsprachen (z.B. Pascal) schon hinter sich hat und sich die größeren Möglichkeiten der C-Programmierung zu nutzte machen will. Deshalb kann man beim C Einsteiger Computer- und Programmiergrundkenntnisse vorraussetzen.

Diesen Weg geht auch der Autor und kann auf diese Weise kurz und bündig die grundlegenden Sprachelemente von C einführen. Dabei kommen mir allerdings die Strukturen zu kurz, die C gerade von anderen modularen Sprachen unterscheiden. Zum Beispiel könnte der Unterschied zwischen Präfix- (++x) und Postfixnotierung (x++) des Inkrementoperators besser beschrieben sein und dessen Einsatzmöglichkeiten erläutert werden. Die Speicherklassen von Variablen werden unnötig früh eingeführt (noch vor den Operatoren) und im Falle der Klassen static und extern sogar irreführend beschrieben.

Neben den üblichen Sprachstrukturen beschäftigt sich der Autor noch mit den Grafikfunktionen von Turbo C++, sowie den Interruptfunktionen. Er beschreibt den Umgang mit der integrierten Entwicklungsumgebung, dem Debugger, dem Library-Utility TLIB, sowie dem Kommandozeilen Compiler TCC. Desweiteren enthält das Buch eine alphabetische Liste der Runtime-Bibliotheksfunktionen mit Kurzbeschreibung.

Ein wenig vermisse ich ein Abgrenzen der Sprachelemente, die im ANSI-C enthalten sind von den Turbo C++ Spezialitäten. Der Autor benutzt die Erweiterungen ganz selbstverständlich ohne Hinweis.

Eine größere Abhandlung über die objekt orientierte Programmierung (OOP) oder den Virtual-Runtime-Object-Oriented-Memory-Manager (VROOM) enthält das Buch nicht. Es gibt nur ein kleines Kapitel zu diesem Thema. Dasselbe gilt auch für das Mixed-Language-Programming Konzept. Wer sich hierfür interessiert, muß auf entsprechende Spezialliteratur zurückgreifen.

Das Buch enthält viele Beispielprogramme. Außerdem sind viele Aufgaben mit dazugehörigen Lösungen im Anhang. Das gibt einem gleich die Möglichkeit sein neues Wissen auszuprobieren. Sämtliche Programme sind auf der mitgelieferten Diskette enthalten.

Zu Empfehlen ist das Buch für C-Anfänger, die nur mit Turbo C++ arbeiten wollen und für den Anfang das "++" noch unbenutzt lassen. Durch die kursartige Gliederung kommt man recht schnell zu eigenen leistungsfähigen Programmen. Wer jedoch einen allgemeinen Einstieg in die Programmiersprache C (ohne "++") sucht, ohne an einen speziellen Compiler gebunden zu sein, dem sei eher die C-Bibel "The C Programming Language" von den C-Erfindern Kernighan und Ritchie empfohlen, die es ja auch in deutscher Sprache gibt.

Alexander Wolf

FLAGSTATUS für den HP 48 SX

von Günter Schapka

Angeregt durch den HP-28-Artikel von Wolfgang Führer in Heft 3-90, S.32, habe ich mein bereits vorher existierendes, aber erheblich umständlicher arbeitendes Programm FSTAT zur Darstellung des Systemflagzustands neu bearbeitet. Es besteht jetzt aus seiner FLG Abfrageroutine für die Flags und einem Grafikobjekt, das vorher mittels Programm PID (gleicher Name) erstellt wurde. Werden nach dem Flagtest alle Felder für die Grafik gerechnet, dann setzt erst einmal ein großes Bildschirmflimmern ein und das Ganze dauert deutlich länger.
Durch den REPL-Befehl ist PID für den HP 48 SX erheblich einfacher. Mit dem Programm FX1 wurde die Grafik erstellt, in FX2 abgespeichert und dann in FSTAT mittels Programm INS (Prisma 3-90, S.37) eingebunden. Ich gehe davon aus, daß mit PID meistens inverse Werte erzeugt werden, deshalb bei positivem Vorzeichen der Platzziffer inverse Darstellung, bei negativem Vorzeichen normale Darstellung. Maximal sind 2*22=154 Positionen, von Null bis 153 möglich.
Zum Aufruf ist der anzuzeigende Flagzustand mittels RCLF in den Stack zu holen. Soll der User-Flagbereich gezeigt werden, dann System- und User-Werte vor Aufruf von FSTAT im Stack mittels der Befehle OBJ→ SWAP ROT ROT →LIST umtauschen.

FSTAT

```
    « RCLF → flags
      « STOF CLLCD
0 3
        FOR x "" 1
16
          FOR y x
16 * y + NEG FS?
→STR y 4 MOD 0 ==
"   " IFT +
          NEXT + +
+ + 1 22 SUB x 2 +
DISP
        NEXT LCD→ {
# 17h # 2h }
GROB 86 39
656771100C5883300000001512
1B1004445808000000222751C
1C5C5310000000424211100444
52280000003232711004C59B10
00000000000000000000000000
00EFB100000EF81000006F81EF
9100000E771000002771EFA100
000E771000006F71E7B100000E
F81000006F91E70100000E7710
00006FE1EFB100000E77100000
67F1EFB100000EF81000002601
EFF100000EFF100000EFF12E81
000002EB1000002E81C5710000
0CD9100000C571E53100000EDA
100000E57166510000066B1000
006E81A76100000A70100000A7
71C77100000CFB100000C7710C
81000000CB1000000C81EFF100
000EFF100000EFF12E9100000E
E8100000EEB1CDE10000066710
00006E91E5F100000A63100000
AEA1268100000C65100000C6B1
E571000000461000000401C571
00000E67100000EEB12E810000
0EE8100000EEB1EFF100000EFF
100000EFF10C81000000C91000
006E81C77100000CFE100000A7
710E710000006F100000C731ED
9100000E581000000651EDE100
000E57100000C561C5F100000C
57100000C5712601000002E810
00002E81EFF100000EFF100000
EFF10000000000000000000000
GXOR →LCD 7 FREEZE
0 WAIT DROP flags
STOF
    »
  »
```

FX1

```
    « CLLCD STD
" 4" 26 PID " 8" 32
PID "12" 38 PID
"20" 48 PID "24" 54
PID "28" 60 PID
"36" 70 PID "40" 76
PID "44" 82 PID
"52" 92 PID "56" 98
PID "60" 104 PID
LCD→ { # 17h # 2h }
"SYSTEM - FLAGS :"
1 →GROB REPL {
# 17h # 2h } {
# 6Ch # 28h } SUB
'FX2' STO
    »
```

PID

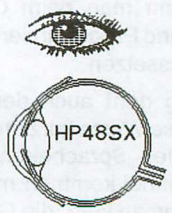
```
    « LCD→ SWAP DUP
      IF 0 <
      THEN 1 SF NEG
      END SWAP ROT
DUP 5 DISP LCD→ {
# 0h # 20h } ROT
SIZE 6 * 1 - R→B 39
R→B 2 →LIST SUB
      IF 1 FS?C
      THEN NEG
      END NEG ROT
DUP 22 MOD 6 * R→B
SWAP 22 / IP 8 *
R→B 2 →LIST SWAP
REPL →LCD
    »
```

Günter Schapka
Rebusgasse 11
D6100 Darmstadt

Der HP48 als Barcodeleser ?

Der HP48SX verfügt über eine Infrarotsende- und Empfangsdiode.
Im Anhang A seines Handbuchs gibt es einen Infrarot-Schleifentest, der offenbar darauf beruht, daß der HP48SX zugleich senden und auch empfangen kann.
Meine Überlegung:
Mit Linsen könnte man das Licht für die Sende- und Empfangsdiode bündeln, um so Barcode lesen zu können. Dazu müßte man wissen, welches Material für diese Linsen geeignet ist, ob auch im infraroten Licht die Kontraste von Thermopapier und PRISMA-Druckfarben erhalten blieben, und wie man per Software die Dioden bedient.
Schließlich muß der Standard für die Form der Barcodes (z.B. wie der EAN-Code der Scanner-Kassen im Supermarkt) und deren Aufbau (Typ, Prüfsumme, Länge etc.) her.
Für Ideen und Wissen wäre ich jedem dankbar, der dazu etwas beitragen könnte, jede Kleinigkeit ist willkommen.

Ein Denkanstoß



Ralf Pfeifer
Rubenstraße 5
D5000 Köln 50

Datenkomprimierung und Archivierung
auf dem HP48SX

Das nun folgende Programmpaket ist eigentlich dreigeteilt:

1. Die Druckersteuerungen aus PRISMA 5-6/90 habe ich ein wenig kürzer gestaltet. Die einzelnen Funktionen werden jetzt aus dem HOME-Directory direkt aufgerufen, d.h. ohne Flags; sie sind als USER-Funktionen direkt nutzbar.
2. Ich habe ein leicht abgeändertes CHECK-Programm beigefügt, das auf dem TIMER-Programm von Ralf Pfeifer aus PRISMA 5-6/90 aufbaut. Hier habe ich in Zeile 11 ein '7 TRNC' eingefügt, da auf diese Art immer eine optisch korrekte Anzeige erfolgt. Es fehlen jetzt die lästigen vorangestellten Nullen.
   Die beiden Programme PRP und PRDIR sind ebenfalls von Ralf Pfeifer.
3. Der große Packen ist eine Arbeitsumgebung, in der Daten gepackt archiviert werden können, um das RAM im Rechner nicht als teure Ablage zu mißbrauchen; dazu später aber mehr.

Zu dem Programm IN(put) ist eigentlich nur zu bemerken, daß es den "Tagged-String" in Ebene 2: und die Eingabekommentierung in der Ebene 1: des Stacks erwartet. Danach wird der eingebaute Input-Befehl derart genutzt, daß für eine nachträgliche Druckanzeige zuerst die Eingabeaufforderung ohne das "Eingeben " steht (sozusagen als Überschrift) und das Tagged-Objekt normal nach PR1 DROP weiter verarbeitet werden kann (z.B. ebenfalls PR1 DTAG ....). Zwar können nach dieser Methode immer nur einzelne Eingabeaufforderungen Schritt für Schritt verarbeitet werden, aber die Programme werden einfacher und übersichtlicher, da die vielen SWAPs und ROTs entfallen können.

Das Programm "plot" benutzt die Schnelligkeit von DRAW (deshalb vorher darauf achten, daß richtig skaliert ist), es sind mehrere Funktionen möglich (vergleiche Handbuch Kapitel für das Plotten).

PRF entspricht dem Fettdruck des HP-Druckers, PR2 stellt die Zeichen etwas höher dar (wie bei PRDIR die Überschriften) und PRK ist der Kleindruck...

Zu den neuen Programmen WS→A (Workspace to Archive) und A→WS (Archive-File to Workspace ist zu sagen, daß sie speziell von mir für eine kombinierte Anwendung von Datenkompressionsprogramm und dem Workspace-Programm entwickelt wurden. Aus diesem Grunde habe ich die Teilprogramme des Komprimierungsteils leicht modifiziert:
In XARC erhält Zeile 1 die lokale Variable wsn (*workspacenamen*), diese Variable wird in Zeile 16-18 des gleichen Programms genutzt, um das vorher definierte Workspace anzuspringen und die Variablen Schritt für Schritt richtig abzuspeichern. Da das Datenkomprimierungsprogramm beim XARCen sequentiell arbeitet, sind die Hin- und Hersprünge notwendig, damit das Programm immer die korrekten Daten verarbeitet.

Im Programm AR sind die Zeilen 27-30 entsprechend angepaßt worden, damit auch die komprimierten Daten im richtigen Speicher landen. Die beigefügten Hilfsvariablen sind im Druck etwas mickrig ausgefallen. Sie sind aber außerordentlich wichtig, d.h. sie müssen unbedingt korrekt terminiert werden. Dies gilt insbesondere für "arck" - eine große Fehlerquelle für das automatische Ent- und Verpacken, wenn nur ein einziger Leerstring falsch steht ....

Die Variable "dict" ist meine Version einer vollständigen Ausnutzung der 127 möglichen Befehle, d.h. es kann jeder seine Version ausprobieren. Es muß nur sichergestellt sein, daß « und » am Anfang stehen.

Das Workspace-Programm

Das Workspace-Programm ist eigentlich eine Entwicklungs-Umgebung, die für jeden einzelnen individuell ausnutzbar ist.
Auf der anderen Seite ist die Programmierung selbst ein gut gelungenes Beispiel für eine saubere und strukturierte Programmierung.
Nun also ein selbst erklärendes Beispiel für künftige Anwendungen; nur das Notwendigste ist im USER-Menü sichtbar, alle Dienstleistungen sind zentral verwendbar und die wichtigsten Variablen sind gegen das unbeabsichtigte Löschen geschützt usw...

Dies war einer der Gründe, warum es mir besonderen Spaß gemacht hat, dieses Programm zusammen mit den ARC-Programmen laufen zu lassen. Die benötigten 9 kByte für beide Programme kommen sehr schnell wieder dem Anwender zugute, da ARC & Co den Rechner aufräumt und eigentlich mehr kBytes zur Verfügung stellt, als es selbst verbraucht. Und der Vorteil einer strukturierten Programmierung ist ja auch nicht von der Hand zu weisen.

Zu der Arbeitsweise kann eigentlich nur wenig gesagt werden, da jeder individuell bestimmt, was aus dem Programm zu machen ist. Aus diesem Grunde erfolgt nur eine Kurzangabe, was die einzelnen Programmteile machen bzw. wobei sie mitwirken.

Eine Reihe von Programmen muß unbedingt im HOME-Directory abgelegt werden (so wie die ARC-Programme auch). Dies sind PUTV, GETV, Purge, CD, SOB, MV, CP und WM.
Dies sind alles Dienstprogramme, die für die Arbeitsräume (Workspaces) benötigt werden. Sie werden nicht vom USER direkt angesprochen und können deshalb relativ kurz abgehandelt werden. Die einzige Ausnahme ist hier WM.

PUTV	ist verantwortlich für das Variablengeschiebes innerhalb der Workspaces
GETV	greift sich zunächst die definierten Variablen und unterstützt u.a. PUTV
Purge	löscht sowohl Variable als auch Directories
CD	ist leider keine neue Platte sondern hilft beim Wechsel der Verzeichnisse
MV	ist an dem Verschiebespiel von PUTV und GETV beteiligt, eine Variable mit MV wird im aufgerufenen Verzeichnis gelöscht und im neuen gespeichert (vergleiche übergeordneten Befehl (MOVE)
CP	Wie MV, jedoch nur kopieren ohne zu löschen (übergeordneter Befehl COPY)
SOB	ermittelt SIZE von einem Verzeichnis (übergeordneter Befehl Size)
WM	bildet aus HOME-Directory die Hauptumgebung des Workspace-Menus (MAIN) [vom USER aus aufzurufen]
$\overline{WS}$	Unterverzeichnis im HOME und in $\overline{WS}$ selbst eine Listenvariable für weitere Unterverzeichnisse, die durch den USER definiert werden; in meinem Fall waren es TEST und ARC (am Anfang eine

Leerliste { })

GUTIL ist eine Listenvariable, die das Menü für MAIN bereithält

SUTIL ist eine Listenvariable für das durch den USER- definierte WRK-Verzeichnis; dies ist das eigentliche Arbeitsverzeichnis der definierten Umgebung

Als Programme in WS sind folgende kurz angesprochen:

MAIN bildet Menü in der Eingangsumgebung des Workspaceprogramms (FUNCTION)

STRT Hilfsprogramm für SMENU, nicht als FUNCTION aufrufbar

SMENU bildet Menü in WRK, Hilfsprogramm, nicht aufrufbar als FUNCTION

LOAD holt Programme aus dem PC (sofern vorhanden) FUNCTION

SAVE speichert diese im PC (FUNCTION)

Size ermittelt den Speicherbedarf des aufgerufenen Workspaces mit allen Hilfsvariablen (daher immer 203 Byte abziehen, um den eigentlichen Datenverbrauch der eigenen Programme zu ermitteln) [vergleiche A→WS in Zeile 21] FUNCTION

BUILD bildet das USER-Workspace mit allen Hilfsvariablen und Unterverzeichnissen. Das neue Workspace muß dann direkt aus MAIN aufgerufen werden (es ist in der Menüleiste zu finden) FUNCTION

CRUSH löscht alle Variablen des definierten Workspace, es muß deshalb als Notbremse betätigt werden (FUNCTION)

DELL Hilfsprogramm, es dient der Löschroutine, es ist nicht aufrufbar

MOVE verlädt Variable in ein neues Workspace und löscht diese im alten (FUNCTION)

COPY wie MOVE, jedoch ohne Löschen

EXEC führt ein definiertes Kommando oder Programm aus Stack 2: mit definiertem Workspace in Stack 1: aus; d.h. man kann auch Programme aus Nebenverzeichnisse ausführen !!, was sonst nicht möglich ist (vergleiche Variablenstruktur in Verzeichnissen, Handbuch HP48) (FUNCTION)

GLOBL macht Variable für aktuelles Verzeichnis verfügbar (FUNCTION)

LOCAL nur für definiertes Workspace verwendbar (FUNCTION)

HIDE versteckt Variable in höhergelegenen Verzeichnissen, so daß diese nicht mehr im USER-Verzeichnis sichtbar sind. Sie können mit LOCAL wieder zurückgeholt werden (FUNCTION)

GLLBL wie GLOBL, zusätzlich erscheint die Variable im USER-Menü aller Verzeichnisse (FUNCTION)

NOLBL löscht die betreffenden Variablen, leider auch die im HOME-Directory, also aufpassen bei Doppelnamen!! (FUNCTION)

SPLBL wie GLOBL, jedoch erscheint die Variable im USER-Menü an erster Stelle, kann also als Startvariable für das WRK-Menü verwendet werden, um die eigene Anwendung zu beginnen... (FUNCTION)

Nun folgen einige Beispiele für die Anwendung der oben beschriebenen Programme. Die FUNCTION-Befehle sind in der Regel einwertig, d.h. 'Name' in den Stack 1: und los geht's...
Als Variablenname ist in der Regel immer Einzelbenennung und Listenform zulässig. Bei MOVE und COPY ist die Reihenfolge 'Name' 'Workspace Ausgang' 'Workspace Ziel' einzuhalten, dann wie gewohnt FUNCTION und ab geht die Post...
EXIT als FUNCTION in MAIN verläßt die Workspaceumgebung und landet im HOME-Verzeichnis. In meinem Beispiel im USER-Menü, um von dort aus wieder weiter zu machen; dies erspart mir den Anblick der vielen Hilfsprogramme.

RESET sollte nur in Verbindung mit der gewünschten Grundeinstellung des Rechners genutzt werden: Einmal alle gewünschten Modi einstellen, dann MAIN drücken und RCLF, diese dann in FLAG abspeichern. Damit holt einen der RESET-Befehl wieder in die "Normalstellung" zurück, dies umso interessanter, wenn man eigene Programme austestet. Es passiert recht häufig, daß der Programm-Code korrekt ist, nur ein Flag war noch falsch gesetzt. Genau genommen sollte natürlich jedes Programm seine Flags richtig vorsetzen, bei einzelnen Routinen ist es da schon schwieriger.

Eine Anmerkung noch zu GLBL:
Diese Routine dient in erster Linie dazu, die eingebauten FUNCTIONs des Rechners in das USER-Menü zu holen. Dies kann aber auch für Programme benutzt werden, vergleiche A→WS etc.. Deshalb erfolgt hier die Eingabe mit Liste !!, da sonst die eingebauten Befehle des Rechners sofort ausgeführt werden. Nach dem Aufruf sind die Befehle jederzeit aus dem USER-Menü heraus ausführbar.

So, das war's an notwendigen Erklärungen. Der eigentliche Spaß sollte beim Ausprobieren und spielerischen Weiterentwickeln des Programms kommen. Jeder kann individuell bestimmen, wie groß die Unterstützung eigener künftiger Programmierarbeit sein wird. In Verbindung mit dem Komprimierprogramm kann auch mehr an Programmen und Daten im Rechner selbst gehalten werden, ohne diesen immer wieder leeren zu müssen.

Ich würde mich freuen, wenn Weiterentwicklungen dieses Programms demnächst im PRISMA erscheinen - oder vielleicht Anmerkungen, wie dieses Programm sonst noch genutzt werden kann. Wie wäre es mit einer automatischen Umformung in Maschinensprache für normal konstruierte Programme ?? Tüftler an die Front ...

Zwei Anmerkungen in eigener Sache:
Mir fällt oft auf, daß im PRISMA veröffentlichte Programme zwar häufig sehr gute Einzellösungen darstellen, jedoch durch den Verzicht auf "sprechende Variablen", d.h. aussagekräfte Variablennamen, nur schwer nachvollziehbar sind. Auch wird häufig keine Grundeinstellung mit angegeben, d.h. die Stellungen RAD oder DEC etc..
Dies ist der eigenen Anbindung an selbst genutzte Programme nicht gerade dienlich. Auch wenn dies eine "Verschwendung von Bytes" darstellt, andernfalls ist eine Verschwendung von Wissen. DANKE !!

Sodann suche ich noch ein Programm für numerische Mathematik und Statistik, insbesondere Regressionsanalyse mit Polynomen und die Tschebischeff-approximation, um hierfür ein größeres Programmpaket zusammenzustellen.

Für alle Astronomen: ich habe ein sehr langes Programmpaket "Sternenalmanach", das ich gerne als Kopie für Interessierte zur Verfügung stelle, also bitte bei mir melden. Ich selbst habe leider kein großes Interesse an Astronomie, das Programm ist aber zu schön, um es im Keller verstauben zu lassen. Vielleicht kann der eine oder andere damit etwas anfangen.

Die Grundversion des Workspaceprogramms stammt von einem Friedrich Schröder, RC International, Copenhagen, Denmark.

Georg Hoppen
Hubertusring 5
4512 Wallenhorst

Hilfsroutinen

```
48SX PRF
   « DUP 27 CHR 253
002  CHR + SWAP + 27 CHR
   + 252 CHR + PR1 CR
004  DROP
   »  Bytes : 84,5
006  Checksum : # 88CCh

48SX PR2
   « DUP 3 →GROB PR1
002  DROP
   »  Bytes : 22,5
004  Checksum : # 842Bh

48SX PRK
   « DUP 1 →GROB PR1
002  DROP
   »  Bytes : 22,5
004  Checksum : # AA30h

48SX IN
   « DUP "
002  eingeben" +
   ROT INPUT OBJ→ SWAP
004  »  Bytes : 39
   Checksum : # 81C8h

48SX CHK
   «
002    IFERR RCL
     THEN TEXT
004    END MEM DROP →
   test
006    « TICKS → t
       « test EVAL
008  TICKS t -
       » 8→R 29491200
010  / →HMS 7 RND 0 HMS+
   STD 7 TRNC →STR
012  "HHMMSSssss" test
   BYTES 'Bytes' →TAG
014  SWAP 'chk∑' →TAG
     »
016  »  Bytes : 172
   Checksum : # A03h

48SX WS→A
   « 0 → o
002    « VARS DUP OBJ→ 1
   SWAP
004      START BYTES 'p'
   STO+ DROP
006      NEXT
   "Name ArcFile ?
008  mit ' ' eingeben,,,
   "
010  { α } INPUT OBJ→ AR
   HOME WS MAIN 100 *
012  p / "%" →TAG
     » TEXT
014  "Workspace löschen
   ?
016  'Name' und CRUSH,,,
   "
018  PROMPT KILL
   »  Bytes : 213,5
020  Checksum : # 22F9h

48SX A→WS
   « CLEAR
002  "Name Workspace ?
   mit ' ' eingeben...
004  "
   { α } INPUT STR→ →
006  ws
       « HOME WS WSU ws
008      IFERR RCL
       THEN UPDIR ws
010  BUILD
         ELSE DROP
012      END WSU ARCH
   WRK 2 MENU
014  "Name ArcFile ?
   mit ' ' eingeben...
016  "
   { α } INPUT STR→
018  DUP ARCL SWAP ws
   XARC HOME WS ws
020  EVAL 1 MENU ws DUP
   Size 203 - 'Bytes'
022  →TAG
     »
024  »  Bytes : 307,5
   Checksum : # 31C8h
```

```
48SX Plot
   « PICT PURGE DRAW
002  DRAX LABEL PRLCD CR
   TEXT
004  »  Bytes : 30
   Checksum : # C7A7h
48SX ARCL
   « RCL → ls
002    « 1 ls SIZE
       FOR i ls i GET
004  + 3
       STEP
006    »
   »  Bytes : 68
008  Checksum : # BE98h
48SX XARC
   « → wsn
002    « → ls af
       « 1 ls SIZE
004      FOR i ls i
   GET → on
006        «
   "XARcing " on →STR
008  + 1 DISP af RCL DUP
   DUP on
010              IF POS
   DUP
012                THEN SWAP
   OVER 1 - GET 3
014  ROLLD 1 + GET 12 +
   arck SWAP GET STR→
016  UPDIR wsn EVAL WRK
   on STO UPDIR ARCH
018  WRK
                ELSE 3
020  DROPN
                END
022            »
             NEXT TEXT
024    »
     »
026  »  Bytes : 263
   Checksum : # F9AEh
48SX AP
   « → ls af
002    « -55 CF af
       IFERR RCL
004    THEN { }
       END 1 ls SIZE
006    FOR i ls i GET
   DUP "Arcing " OVER
008  →STR + 1 DISP
       IFERR RCL
010    THEN DROP
       ELSE DUP TYPE
012  → t
           « arck t 1
014  + GET STR→ → on ob
             « DUP
016            IF on
   POS DUP
018            THEN
   SWAP OVER 1 - ob
020  PUT SWAP 1 + t PUT
               ELSE
022  DROP ob + on + t +
                END
024          »
           »
026      END
       NEXT HOME WS
028  WSU ARCH WRK af STO
   TEXT af RCL BYTES
030  SWAP DROP
     »
032  »  Bytes : 344,5
   Checksum : # DD66h
48SX XMASH
   « DUP t 1 SUB NUM
002  "(" "" IFTE → o t
     « o ")" POS 1 + o
004  SIZE
       FOR i t i 1 + o
006  1 i SUB NUM 2 / DUP
   4 ROLLD i + ,5 +
008      FOR j mlex o
   j IP DUP SUB NUM
010  DUP 16 / IP
           IF j FP ,5
012  SAME   //
           THEN 16 * -
014        ELSE SWAP
   DROP
016        END DUP SUB
   + ,5
```

```
018      STEP STR→
   SWAP DUP
020      IF FP ,5 SAME
         THEN ,5 +
022      END 1 +
       STEP "(" o 2 O
024  ")" POS SUB + STR→
   →ARRY
026    »
     »  Bytes : 374
028  Checksum : # 5B19h
```

Archivierung in Verbindung mit Workspace + WS →A, A →WS

```
48SX MASH
   « RCLF → flag
002    « 1 OVER TYPE
         IF 4 SAME
004      THEN SF
         ELSE CF
006      END ARRY→ DUP
   →STR 3 OVER SIZE 2
008  - SUB "}" + 1 FS?
   CHR SWAP + DEPTH
010  ROLLD LIST→
         IF 2 SAME
012      THEN *
         END 1
014      FOR j j ROLL
   →STR
016        IF 1 FS?
           THEN 2 OVER
018  SIZE 1 - SUB
           END DUP SIZE
020  DUP CHR 3 ROLLD → o
   z
022          « 1 z
               FOR i mlex
024  o i i SUB POS 16 *
               IF z i >
026            THEN mlex
   o i 1 + DUP SUB POS
028  +
               END CHR +
030  2
           STEP
032        » DEPTH ROLL
   SWAP + DEPTH ROLLD
034  -1
       STEP DEPTH ROLL
036  flag STOF
     »
038  »  Bytes : 357,5
   Checksum : # AC1Dh
48SX XPAK
   « → ob
002    « "«" 1 ob SIZE
       FOR i " " + ob
004  i DUP SUB NUM DUP
         IF 127 >
006      THEN ob SWAP
   128 - i 1 + SWAP
008  DUP 1 + 5 ROLLD i +
   SUB + SWAP
010      ELSE dict
   SWAP GET + 1
012      END
       STEP STR→
014    »
     »  Bytes : 190,5
016  Checksum : # E68Ch

48SX PAK
   « →STR { " " } 10
002  CHR + → obj eoo
       « "" DUP 1 obj
004  SIZE
         FOR i i 3 ROLLD
006  obj i i SUB eoo
   OVER
008        IF POS DUP
           THEN
010          IF 2 SAME
             THEN 4 ROLL
012            DO 1 +
               UNTIL obj
014  OVER DUP SUB " " ≠
             END 1 - 4
016  ROLLD
         END DROP
018  dict OVER
           IF POS DUP
020        THEN CHR
   SWAP DROP +
022        ELSE DROP
```

```
    DUP SIZE 128 + CHR
024 SWAP + +
          END ""
026       ELSE DROP +
          END 3 ROLL i
028 - 1 +
       STEP DROP
030    WHILE DUP DUP
    SIZE DUP SUB NUM 2
032 SAME
       REPEAT 1 OVER
034 SIZE 1 - SUB
       END 2 OVER SIZE
036 SUB
    »
038 »  Bytes : 382
    Checksum : # A88Ah
```

Liste der Verzeichnisinhalte

```
HOME DIR :
   08 WS→A
   08 PRF
   08 PRP
   08 PRK
   08 IN
   08 CHK
   08 Plot
   08 A→WS
   05 arck
   08 ARCL
   08 XARC
   08 AR
   02 mlex
   08 XMASH
   08 MASH
   05 dict
   08 XPAK
   08 PAK
   08 PRP
   08 PRDIR
   05 CST
   05 PRTPAR
   15 WS
      08 TEST
      08 ARCH
      05 FLAG
      08 SPLBL
      08 NOLBL
      08 GLLBL
      08 HIDE
      08 LOCAL
      08 GLOBL
      08 EXEC
      08 COPY
      08 MOVE
      08 DELL
      08 CRUSH
      08 BUILD
      08 Size
      08 SAVE
      08 LOAD
      08 SMENU
      08 STRT
      08 MAIN
      05 SUTIL
      05 GUTIL
      05 WS
      05 CST
      15 WSU
         15 TEST
            05 CST
            05 SPEC
            15 WRK
               (Kein Eintrag
)
         15 ARCH
            05 CST
            05 SPEC
            15 WRK
               (Kein Eintrag
)
         05 WS
         05 GLOB
   08 PUTV
   08 GETV
   08 Purge
   08 CD
   08 SDB
   08 MV
   08 CP
   08 WM
```

WS →A	speichert Workspace im Archiv
PRF - plot	Druck, Eingabe und Plotfunktionen
arck - PAK	Archivierungsprogramme
PRP;PRDIR	Programme aus PRISMA

noch in HOME: PUTV - WM (Hilfsroutinen für Workspace)

Subdirectories:
WS
Test, Archiv } weitere Subdirectories
Flag — eigene Flagstellung (Variable)
SPLBL-MAIN — Hilfsprogramme für Workspace
SUTIL-CST — Hilfsvariable für Workspace

In Test bzw. Archiv:
Spec — Hilfsvariable
WRK — "Arbeits"-Subdirectory
GLOB — gemeinsame Hilfsvariable für Funktionen im CST-Menü

Hilfsvaliablen für ARC-Programm

```
6: 'arck'
5: { "" "" "" "MASH"
"MASH" "1 →LIST" "→STR"
"" "PAK" "→STR" "" "" ""
"" "XMASH" "XMASH" ""
"STR→" "" "XPAK" "STR→"
"" }
4: 'mlex'
3: "0123456789;,E-"
2: 'dict'
1: { "«" "»" "+" "DUP"
"STO" "END" "SWAP" "→"
"THEN" "IF" "NEXT" "FOR"
"GET" "-" "DROP" "SIZE"
"→STR" "*" "SUB" "(" ")"
"PURGE" "SAME" "ELSE"
"PATH" "ROL" "STR→"
"CHR" "TYPE" "NUM" "PUT"
"→LIST" "EVAL" "/" "^"
"R→B" "DISP" "POS"
"MENU" "NOT" ">" "DO"
"UNTIL" "START" "HOME"
"PIXEL" "[" "]" "CLLCD"
"STO+" "R→C" "ROM" "B→R"
"S@" "ROT" "PICK" "SF"
"REPEAT" "WHILE" "ROLL"
"ROLLD" "OR" "IP" "%"
"MOD" "DROP2" "AND"
"FS?" "#" "OVER" "IFERR"
"KEY" "LIST→" "STOF"
"CRDIR" "VARS" "FC?"
"PMIN" "PMAX" "RAND"
"SIN" "COS" "C→R" "FP"
"<" "≤" "TEXT" "DEPTH"
"DRAX" "BEEP" "CLEAR"
"RCLF" "HALT" "STO-"
"CF" "≠" "INV" "SYSEVAL"
"STWS" "XOR" "PR1"
"INPUT" "PROMPT" "PICT"
"LABEL" "DRLCD" "DRAX"
"Σ+" "STEQ" "STOΣ"
"RCLΣ" "TRN" "ARRY→"
"OBJ→" "→ARRY" "→TAG"
"DROPN" "LINFIT"
"EXPFIT" "PWRFIT"
"LOGFIT" "LR" "RSD"
"DOT" "REPL" "ABS"
"DTAG" }
```

Workspace-Programme in HOME

```
485X PUTV
   « OBJ→ 1
002   START SWAP STO -2
      STEP
004 »  Bytes : 27,5
   Checksum : # 4858h

485X GETV
   « { } + DUP SIZE →
002 1 s
      «
004     IF s
        THEN 1 s
006       FOR i 1 i GET
   DUP ROL
008       NEXT
        END s 2 * →LIST
010   »
   »  Bytes : 101
012 Checksum : # 98EEh

485X Purge
   « { } + DUP SIZE →
002 1 s
      « ROLF -55 CF
004     IF s
        THEN 1 s
006       FOR i 1 i GET
            IFERR PURGE
008         THEN PGDIR
            END
010       NEXT
        END STOF
012   »
   »  Bytes : 114,5
014 Checksum : # 4F09h

485X CD
   « 1
002   DO GETI EVAL
      UNTIL -64 FS?
004   END DROP2
   »  Bytes : 40,5
006 Checksum : # BE12h

485X SDB
   « { } + DUP SIZE →
002 1 s
      « 0
004     IF s
        THEN 1 s
006       FOR i 1 i GET
   BYTES SWAP DROP +
008       NEXT
        END
010   »
   »  Bytes : 96,5
012 Checksum : # 8B96h

485X MV
   « PATH 4 ROLLD SWAP
002 CD SWAP DUP GETV
   SWAP Purge SWAP CD
004 PUTV CD
   »  Bytes : 70
006 Checksum : # 9846h

485X CP
   « PATH 4 ROLLD SWAP
002 CD SWAP GETV SWAP
   CD PUTV CD
004 »  Bytes : 56,5
   Checksum : # 2A01h

485X WM
   « WS MAIN
002 »  Bytes : 23
   Checksum : # 61ADh
```

Hilfsvariable für Workspace

```
6: 'SUTIL'
5: { MAIN { EXIT
« HOME 1 MENU
» } SPLBL GLLBL NOLBL
HIDE LOCAL GLOBL COPY
MOVE EXEC }
4: 'GUTIL'
3: { { EXIT
« HOME 1 MENU
» } { RESET
« CLEAR FLAG STOF
» } BUILD CRUSH Size
SAVE LOAD PRP }
2: 'WS'
1: { ARCH TEST }
```

Programme im WS-Verzeichnis

```
485X NOLBL
     « → l
002    « PATH WRK UPDIR
     l 'SPEC' DELL PATH
004  UPDIR l 'GLOB' DELL
     CD SMENU CD
006    »
     »  Bytes : 107
008  Checksum : # 156Ch

485X GLLBL
     « GLOB + PATH SWAP
002  WSU 'GLOB' STO DUP
     CD WRK UPDIR SMENU
004  CD
     »  Bytes : 77,5
006  Checksum : # E0F6h


485X HIDE
     « PATH DUP 1 4 SUB
002  MV
     »  Bytes : 28
004  Checksum : # EC40h

485X LOCAL
     « DUP GETV SWAP
002  PATH DUP 1 4 SUB CD
     OVER PURGE UPDIR
004  SWAP PURGE CD PUTV
     »  Bytes : 66
006  Checksum : # 9F0Bh

485X GLOBL
     « PATH DUP 1 3 SUB
002  MV
     »  Bytes : 28
004  Checksum : # 218Dh


485X EXEC
     « PATH → p
002    « EVAL WRK EVAL p
     CD
004    »
     »  Bytes : 46
006  Checksum : # 1264h

485X COPY
     « PATH SWAP OVER 1
002  3 SUB SWAP + 'WRK'
     + CD
004  »  Bytes : 49,5
     Checksum : # E1D6h

485X MOVE
     « PATH SWAP OVER 1
002  3 SUB SWAP + 'WRK'
     + MV
004  »  Bytes : 49,5
     Checksum : # 7F0Dh


485X DELL
     « SWAP { } + → ln v
002    « 1 v SIZE
       FOR j ln RCL
004  DUP v j GET POS DUP
     1 - 3 PICK SWAP 1
006  SWAP SUB SWAP 1 +
     ROT SWAP OVER SIZE
008  SUB + ln STO
       NEXT
010    »
     »  Bytes : 134
012  Checksum : # ED0Ch

485X CRUSH
     « HOME WS DUP →STR
002  ",REALY(Y/N) ?" + (
     α ) INPUT
004    IF "Y" ==
       THEN
006      IF WS OVER POS
         THEN DUP PURGE
008  DUP 'WS' DELL WSU
     Purge MAIN
010      END
       END
012  »  Bytes : 147,5
     Checksum : # 542h
```

```
485X SPLBL
     « SPEC + PATH SWAP
002  WRK UPDIR 'SPEC'
     STO SMENU CD
004  »  Bytes : 63
     Checksum : # FF8Bh

485X BUILD
     « → n
002    «
         IF WS n POS
004      THEN
     "Name used" DOERR
006      ELSE 'WS' DUP
     EVAL n + SWAP STO
008  "«'" n + "' STRT" +
     OBJ→ n STO WSU n
010  DUP CRDIR EVAL
     'WRK' CRDIR { }
012  SPEC STO MAIN
       END
014    »
     »  Bytes : 190,5
016  Checksum : # 7087h

485X Size
     « HOME WS
002    IF WS OVER POS
       THEN WSU SOB
004    END MAIN
     »  Bytes : 61,5
006  Checksum : # 5400h

485X SAVE
     « HOME WS
002    IF WS OVER POS
       THEN WSU
004      IFERR SEND
         THEN
006      END
       END MAIN
008  »  Bytes : 70
     Checksum : # 662Fh

485X LOAD
     « → n
002    «
         IF WS n POS
004      THEN
     "Name used" DOERR
006      ELSE WSU n
           IFERR KGET
008        THEN
           ELSE UPDIR
010  'WS' DUP EVAL n +
     SWAP STO "«'" n +
012  "' STRT" + OBJ→ n
     STO
014        END MAIN
         END
016    »
     »  Bytes : 179
018  Checksum : # 450Fh

485X SMENU
     « SPEC { { } } +
002  GLOB + { { } } +
     SUTIL + MENU WRK
004  »  Bytes : 72,5
     Checksum : # 461h

485X STRT
     « HOME WS WSU EVAL
002  VARS
       IF 'RUN' POS
004    THEN RUN
       ELSE SMENU
006    END
     »  Bytes : 68,5
008  Checksum : # 61DCh

485X MAIN
     « HOME WS WS { { }
002  } + GUTIL + MENU
     »  Bytes : 49,5
004  Checksum : # E70Dh
```

Beispiel für WRK-Umgebung

```
{ { } CHK A→WS WS→A PRP
{ } MAIN { EXIT
« HOME 1 MENU
» } SPLBL GLLBL NOLBL
HIDE LOCAL GLOBL COPY
MOVE EXEC }
```

Hier: Benutzung des CHK-Programmes für Laufzeit und Bytes; WS ›A und A ›WS, um speicherfreundlich zu arbeiten.
PRP, um fertige Programme besser dokumentieren zu können.

Customize your HP28

W.A.C. Mier-Jedrzejowicz

Dieses Buch, das von einigen Programmautoren als Referenz bei Programmen auf Maschinensprachenebene benutzt wird, einige bezeichnen es auch als die Bibel der HP28-Programmierer, ist anscheinend weltweit vergriffen. Der Verlag teilte mit, daß es vorläufig nicht wieder aufgelegt wird.
Aus diesem Grund haben wir uns entschieden eine Kopie des Buches zur Ansicht bereit zu halten.
Man kann diese Kopie erhalten, indem man einer kurzen Notiz 20.-DM als Schutzgebühr beilegt (für den Fall, daß jemand die Kopie behält) und diesen an die Clubadresse schickt.
Bitte nur das Postfach benutzen, da sich die Adresse eventuell ändert !

mm

Schutz vor Programmverlust

beim HP48SX

Bei meiner Beschäftigung mit dem HP48SX konnte ich mich nur auf die Informationen stützen, die in den Handbüchern zu finden waren. Dabei stieß ich darauf, daß es scheinbar keinen Bereich zu geben scheint, der sich schreibschützen läßt.

Folgender Fall passierte mir vor kurzem:
Mit Stolz wird die Variable PGN mit einem neuen Programm belegt; alles scheint in bester Ordnung zu sein.
Eines Tages wird vor der Menü-Taste, die mit PGN belegt ist, versehentlich "left-shift" gedrückt. PGN wird mit dem Inhalt von Level 1 überschrieben, das Programm ist verloren...

Es ist nun deshalb nicht nötig, sich gleich eine RAM-Karte zu kaufen. Vielmehr läßt sich das Programm in Port 0 ablegen, wie auf der Seite 647 des englischen Handbuchs ganz unten fast verschämt bemerkt ist.

Dr. G. Heilmann
Obernhofer Straße 15
5408 Seelbach

Bug Report

Die folgende Aufstellung ist eine sinngemäße Übersetzung von verschiedenen Berichten aus Mailboxen. Die Bugs sind für die Versionen A-D gültig, wenn auch i.d.R. nur die Version D angesprochen wird.

Von der Niederländischen USER-GROUP stammt die folgende Meldung. Doch zuvor noch die Anschrift dieser Gruppe. Sie ist qualitativ am oberen Level der aufgeführten Programmpakete, die ich in naher Zukunft im CCD vorstellen möchte und vielleicht möchte der CCD direkt mit dieser Gruppe in Kontakt treten.

STORC p.a. Christ van Willegen
Dordognelaan 45
5627 HB Eindhoven
The Nederlands

Der Bugbericht stammt von Eric Toonen (Djept-Zuid 6, 5502 RP Veldhoven, The Nederlands).

1 Im EquationWriter wird der Vergleich xy nicht richtig behandelt: Die Werte werden vertauscht.

2 Die ARCHIVE Funktion führt zu einem Crash wenn die Systemuhr im Display angezeigt wird (betrifft Datenübertragung zum PC).

3 SST-> wirkt wie RCL auf ein Verzeichnis im DEBUGMODUS (Abhilfe: Vorher immer mit NEXT überprüfen, welche Schritte folgen werden und bei Verzeichnissen die SST Funktion benutzen).

4 Das Aufrufen von undefinierten lokalen Variablen funktioniert nicht richtig: Zum Vergleich das folgende Kurzprogramm:

1 2 « -> x « « -> y 'x' » » » EVAL EVAL
Diese Folge hinterläßt ein 'External' im Stack nach der Meldung "Undefined Local Name". (28er und 48er)

5 Wenn die COMMAND-LINE aktiviert wird (ohne PRG entry mode) und dann über PTYPE eine Plotfunktion ausgewählt wird, wird die COMMAND-Zeile ausgeführt, aber der Befehl im Stack hinterlassen. Es kann nun eine neue Befehlszeile unterhalb der COMMAND-LINE gestartet werden.

6 Es kann kein GROB der Größe 0*0 in die COMMAND-LINE geladen werden, es sei denn, es ist das letzte Objekt in der Befehlszeile (Abhilfe: #0 #0 BLANK).

7 Der Rechner kommt in eine Endloschleife, wenn eine Unit in sich selbst definiert wird (ON und C gleichzeitig drücken um den Rechner "zurückzuholen").

8.1 Der Parser funktioniert nicht richtig. Bei Verzeichnissen können diese ebenfalls sich selbständig selbst aufrufen, wenn eine Variable in sich selbst definiert wird (Abhilfe wie unter 7).

8.2 Sie können eine Variable verstekken, wenn Sie ein Verzeichnisobjekt editieren und einer Variablen den gleichen Namen geben. Dann haben Sie ein Verzeichnis, das Variablen des gleichen Namens enthält. Es kann nur die erste Variable evaluiert werden, recalled oder mit STO verändert werden. Es sind zwar Labels für alle Einträge (entirie) vorhanden, aber lediglich das erste bestimmt, ob ein Verzeichnisbalken über den Labels angezeigt wird.

8.3 Analog zu 8.2 erscheint die erste Variable stets im Plotkatalog (selbst wenn sie nicht das Verzeichnis darstellt).

9 Wenn Sie ein Verzeichnis aufrufen, dann kann dieses zwar bearbeitet werden, das Ergebnis kann jedoch nicht wieder zurückgespeichert werden.

10 Wenn Sie einen beliebigen Tastenbefehl ausführen und danach sofort die ON-Taste benutzen, landen sie wieder im HOME Verzeichnis. Alle Flags bleiben erhalten, selbst die Befehlszeile, falls sie gerade aktiviert war.

11 Wenn Sie eine Gleichung im EquationWriter bearbeiten mit "implicit() off", danach wieder "implicit on" schalten und danach einen Rücksprung veranlassen (Korrektur), wird die bis dahin eingegebene Gleichung verändert. Sie kann nicht wieder hergestellt werden.

12 Einige KERMIT Befehle haben ein CR/-Befehl am Ende. Dies wird als "_" auf dem Bildschirm dargestellt. Es erscheint eine Meldung "Transfer failed" und "interrupted".

13 Dies ist nicht wirklich ein Bug, aber es ist entäuschend. Wenn Sie den Befehl CLUSR eintippen, wird er als CLVAR dargestellt (identisch mit 28er Befehl).

14 Die Befehle STO und PURGE haben nicht die korrekte interne Adressierung. Sie benutzen die LASTARG Funktion, diese wird entsprechend verändert und zeigt nicht mehr die korrekten Werte an.

14.1 Wenn STO oder PURGE mit zuwenig Argumenten aufgerufen wird, erhalten Sie eine Fehlermeldung "STO Error" oder "PURGE Error". Abhilfe schafft hier die Zuweisung der Befehle im USER Modus (STO 32 PURGE 54.2)STOKEYS.

14.2 Beta ENTER hinterläßt einen Leerstring im Display, wenn Sie die STO oder PURGE Taste aufrufen. (Abhilfe wie unter 14.1)

15 Die <-MATCH und MATCH-> Funktion überprüft keine Gleichungen oder Formeln. Dies ist der Hintergrund dafür, warum überhaupt symbolisch auf dem Rechner gearbeitet werden kann. Aber dies führt auch dazu, daß falsche Ergebnisse angezeigt werden. Beispiel: 'A=A'Listenklammer 'A' '7=8' Listenklammer <-MATCH ergibt das Ergebnis '7=8=7=8'. Dies führt zu einem völlig korrekten Syntaxerror. Anm.: Auch bei der WHILE Funktion scheint der gleiche Fehler aufzutreten. Beispiel: '1=7' DEF führt zu unterschiedlichen Ergebnissen: 'N' {N 1} /=falsch '1' und 'N*1} /richtig 7.

16 Wenn das USER Menü das aktuelle Menü ist und ein Verzeichnis aufgerufen wird, das keine Variable 'CST' enthält, wird ein Systemstop verursacht. Anm. Bei mir (Version C) wird die alte Bezeichnung von CST beibehalten, ohne das ein Systemstop verursacht wird. Ein unbeabsichtigtes Tippen der Menüleiste führt dann zu der Meldung 'Undefined Name' sofern es sich um keine Funktion aus dem Home Verzeichnis handelt.

17 TRNC und RND erlauben auch für einen ARRY in Level 2 und einem Symbol in Level 1 symbolische Behandlung. Beispiel: [2 3]; '7/8' RND ergibt 'RND(UNKNOWN,7/8)'

Auf diesen Bugreport hin hat es eine offizielle Antwort von dem Chefdesigner des HP48 Bill Wickes mit folgendem Inhalt gegeben:

"Es gibt Irritationen über die neue Chipversion"1E". Die "1" ist aus Unachtsamkeit mit in die Darstellung übernommen worden. Offiziell ist nur die E-Version gemeint gewesen (auch die anderen Versionen heißen bei HP 1A, 1B usw.)."

Soviel zu der neuen Version E, die die oben festgestellten Bugs behoben hat - oder auch nicht.

Zu 1. Im EquationWriter wird der Vergleich xy nicht richtig behandelt: Die Werte werden vertauscht.

Ist behoben worden.

Zu 2. Crash mit Uhr im Display während PC Übertragung : Behoben

Zu 3. Unveränderte Handhabung von SST- und SST, da zwei verschiedene Funktionen beabsichtigt sind ohne daß darauf im Handbuch hingewiesen wird.

Zu 4. 'Undefined Label' nicht behoben. Leider ein Fehler in der Funktion -, er habe keinerlei praktische Bedeutung und wird deshalb nicht behoben (es wird nur vergessen den Pointer auf Fehlerbehebung zu setzen).

Zu 5. 'Commandline mit Plottype' nicht behoben - nur Kosmetik (0-Ton HP).

Zu 6. GROB 0*0 nicht behoben, da die vorgeschlagene Abhilfe einfacher ist.

Zu 7. Endlosschleife nicht behoben (0-Ton, Sowas tut man einfach nicht).

Zu 8.1. Kein Bug, es ist einfach nicht wünschenswert, daß der Rechner diese Eigenschaft besitzt, da die Fehlerquellen zu hoch sind.

Zu 8.2. und Zu 8.3. Kein Bug, niemand hat gesagt, daß dies nicht geht.

Zu 9. Kein Bug. Die Generalregel hat Vorrang: Nichts darf in ein angelegtes Verzeichnis gespeichert werden (non empty directory) ohne vorher zu evaluieren.

Zu 10. Tastenprobleme mit ON sind behoben.

Zu 11. Wickes sagt weiterhin dies ist kein Bug.

Zu 12. Kermit Probleme sind nicht behoben.

Zu 13. CLUSR ist kein Bug sondern Kompatibilität zum 28er

Zu 14. STO/PURGE ist kein Bug sondern eine Funktionsmöglichkeit des Rechners.

Zu 15. MATCH Probleme ist auch kein Bug. Es verhindert, daß alle Ausdrücke durch andere ersetzt werden (... und das ist doch nicht erwünscht, oder ???).

Zu 16. CST Probleme - kein Bug. Es entspricht einem RESET Befehl und braucht nicht behoben zu werden (vgl. 10).

Zu 17. TRNC und RND Problem ist behoben worden.

Wichtig ist noch die Warnung vor dem ABS, ARC und SIGN Befehlen im Matr. Menü

Hier werden die Werte von Polar bzw. rechtwinkligen Koordinaten vertauscht:

polar (A; winkelzeichen B) - "P-R" ergibt

'B*cosA)-i*B*sin(A)' anstelle von

'A*cos(B)+i*A*sin(B)'

Einfach die Werte falschrum eingeben, um richtige Werte zu erhalten schafft hier Abhilfe, wenn Rechnerfunktionen genutzt werden sollen oder eigene korrekte USER Funktionen ausführen.

Georg Hoppen

HP Solve Library Application Card

Manch einem wird die Fehlermeldung "EQ: { 'did-not-parse..." einen Strich durch die Rechnung gemacht haben, die er gerade vornehmen wollte, obwohl alles syntaktisch korrekt war.

Die Ursache dieser Meldung ist sehr einfach: Für einige (ziemlich viele) Gleichungen ist es erforderlich, da das amerikanische Dezimaltrennzeichen verwendet wird.

1,050,001.04 = amerikanisches Dezimal-Trennzeichen = "."

1.050.001,04 = deutsches Dezimal-Trennzeichen = ","

Betroffen sind folgende Gleichungen:

1 Stützen und Träger
1.3 Einfache Druchbiegung
1.4 Einfacher Biegewinkel
1.5 Einfaches Moment
1.6 Einfache Abscherung
1.7 Druchbiegung
1.8 Biegewinkel
1.9 Biegemoment
1.10 Abscherung

3 Flüssigkeiten
3.4 Strömung in kreisförmigen Rohren

5 Gase
5.6 Zustandsgleichung für reale Gase
5.7 Zustandsänderung realer Gase

6 Wärmeübertragung
6.1 Wärme
6.2 Thermische Ausdehnung
6.3 Wärmeleitung
6.4 Konvektion
6.5 Wärmeleitung+Konvektion
6.6 Strahlung eines schwarzen Körpers

7 Magnetismus
7.1 Gerade Leiter

13 Halbleiterbauelemente
13.2 NMOS-Transistoren
13.3 Bipolartransistoren
13.4 JFET's

Bemessungshilfen im Stahlbeton

für den 48SX

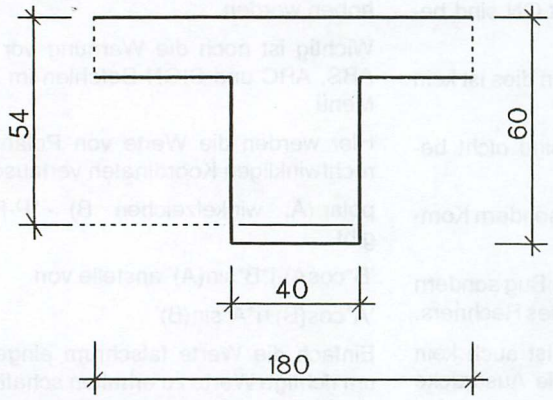
Das Programm "BBEM" führt für Stahlbetontragwerke die Biegebemessung unter reiner Biegung wie auch unter Längskraft mit großer Ausmitte nach dem *kh-Verfahren* auf Grundlage der DIN 1045 durch.
Ermittelt werden die Bewehrung auf der Zugseite (As2) und, falls erforderlich werdend, die Druckbewehrung (As1).
Das Programm verarbeitet die Betongüten B15-B55 sowie die Stahlgüten BSt I, III und IV, wobei allerdings keine Überprüfung auf fehlerhafte Eingaben erfolgt. Nach der weiteren Eingabe der Abmessungen bm, d0 und h (Bild 1) im Programmteil "GEO" sowie der Schnittgrößen M und N ("LOAD") erfolgt die Eingabe der Beiwerte kh, kx, kz, ks und der Bewehrung As2 (bzw. As1, falls erforderlich).

Auf Wunsch kann hieran eine *Schubbemessung* angeschlossen werden. Nach der Eingabe von b0 ("STEG") und der Querkraft Q ("QUER") ermittelt das Programm den Schubbereich, die Schubspannung $\tau 0$ in MN/m², die Bemessungspannung τ, den erforderlichen Bügelquerschnitt (cm²/m) und den zulässigen Höchstabstand e-B der Bügel in Balkenlängsrichtung nach DIN 1045 in cm.
Wird Q negativ eingegeben, so erfolgt die Schubbemessung mit dem zuvor aus der Biegebemessung ermittelten inneren Hebelarm kz * h; andernfalls (Q > 0) wird kz = 0.875 angesetzt.
Sollte $\tau 0$ größer werden als $\tau 3$, so erfolgt eine entsprechende Warnung, ebenso, wenn d0 kleiner als 30cm im Schubbereich 3 ist.
Neuberechnungen mit geänderten Werten sind durch Aufruf der jeweiligen oben angegebenen Programmteile möglich.

Hinweis:
Falls nötig können die Dehnungen ϵ-b1 und ϵ-s2 nach der Bemessung durch Aufruf der Variablen "G" (E-b1) und "P" (E-s2) angezeigt werden.

Eingabe der Normalkraft N als Druckkraft negativ !

Rechenbeispiel:



(Alle Maße in mm)

Baustoffe: Beton B25
Betonstahl Bst IV (4)

Abmessungen: bm = 1.8m
d0 = 0.6m
h = 0.54m
b0 = 0.4m

Schnittgrößen: M = 487kNm
N = 0
Q = 362kN

Im folgenden bedeuten:

▼ ≙ "Cursor down"-Taste

↵ ≙ ENTER

Nach dem Eingeben des Programms muß man für den Aufruf des Variablenmenüs die Taste VAR drücken.

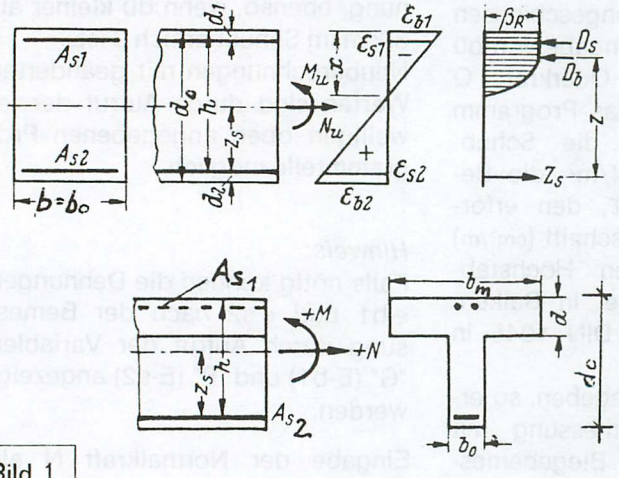


Bild 1

Anzeige	Eingabe/Tastenfolge
	BBEM
Baustoffe Beton B: Stahl Bst:	25 ▼ 4 ↵
Geometrie bm: d0: h:	1.8 ▼ .6 ▼ .54 ↵
Belastung M: N:	487 ▼ 0 ↵
kh: 3.283 kx: 0.200 kz: 0.929 ks: 3.769 As1: 0.000 As2: 33.990	Fortsetzung mit CONT
Stegbreite b0:	.4 ↵
Querkraft Q:	362 ↵
Bereich 3.000 τ0: 1.915 τ : 1.915 as-B: 26.815 e-B: 15.000	

(Änderung mit STEG auf b0 = 0.45m

und Q = 362 ergibt: Bereich 2.000;

τ0 = 1.703; τ = 1.610; as-B = 25.363;

e-B = 20.000)

(schwarze Tasten mit invertierter Schrift sind Softkeys)

```
BBEM
« " Baustoffe" {
":Beton    B:
:Stahl Bst: "
( 1 0 ) } INPUT OBJ→ → y
  «
    CASE 'y==4'
      THEN 500 'BS' STO
      END 'y==3'
      THEN 420 'BS' STO
      END 220 'BS' STO
    END
  » → y
  «
    CASE 'y==25'
      THEN 17.5 .75 1.8
3 LIM
      END 'y==15'
      THEN 10.5 .5 1.2 2
LIM
      END 'y==35'
      THEN 23 1 2.4 4
LIM
      END 'y==45'
      THEN 27 1.1 2.7
4.5 LIM
      END 30 1.25 3 5
LIM
    END
  » GEO
»

GEO
« " Geometrie  [m]" {
":bm:
:do:
:h : " { 1
0 } } INPUT OBJ→ 'H' STO
'D' STO 'B' STO LOAD
»

LOAD
« " Belastung  [kNm,kN]"
{ ":M:
:N: " { 1 0 } }
INPUT OBJ→ 'N' STO 'M'
STO 'N*(D/2-H)' EVAL 'M'
STO+ 0 'C' STO '√(B*
10000/M)*H' EVAL 'K' STO

'H^2*B*BR*193.3' EVAL
'Y' STO
  IF 'M≥Y'
  THEN M Y - 'C' STO Y
'M' STO
  END LB0
»

STEG
« "Stegbreite  [m]" {
":bo: " { 1 0 } } INPUT
OBJ→ 'BO' STO QUER
»

QUER
« "Querkraft  [kN]" {
":Q: " { 1 0 } } INPUT
OBJ→ 'Q' STO
  IF 'Q<0'
  THEN '-Q/H/BO/Z/1000'
EVAL
  ELSE 'Q/H/BO/875' EVAL
  END DUP 'T0' STO → y
  «
    IF 'y≤T1'
    THEN 1 'I' STO LB6
    ELSE 2.5 'I' STO
      IF 'y≤T2'
      THEN LB7
      ELSE
        IF 'y≤T3'
        THEN LB8
        ELSE LB11
        END
      END
    END
  »
»


LIM
« 'T3' STO 'T2' STO 'T1'
STO 'BR' STO
»
```

```
LB0
« 'M/(H^2*B*BR*1000)'
EVAL 'E' STO 'E^3*9.282+
E^2*.728+√E*.295' EVAL
'F' STO
  IF 'E<.097'
  THEN LB1
  ELSE
    IF 'E≥.1578'
    THEN 'F/.4159' EVAL
'X' STO '3.5/X-3.5' EVAL
'P' STO -3.5 'G' STO LB3
    ELSE LB2
    END
  END
»

LB1
« 'F/(E*.41+.335)' EVAL
'X' STO LB4
»

LB2
« '1-√((-52.5*E+15)/19.4
)' EVAL 'X' STO LB4
»

LB3
« 1 F - 'Z' STO 'C*17.5/
(BS*(2*H-D))' EVAL 'Q'
STO '(M/Z/H+N)*17.5/BS+Q
' EVAL 'U' STO
  IF 'Q>U'
  THEN "As1 > As2 !"
HALT
  END '1750/BS/Z' EVAL
'S' STO 'I=1' DEFINE
CLLCD K "kh : " LB5 X
"kx : " LB5 Z "kz : "
LB5 S "ks : " LB5 'I'
INCR DROP Q "As1: " LB5
U "As2: " LB5 3 FREEZE
HALT STEG
»

LB4
« '5*X/(X-1)' EVAL 'G'
STO 5 'P' STO LB3
»

LB5
« SWAP + I DISP 'I' INCR
DROP
»

LB6
« 1 'J' STO T0 DUP 'T'
STO .4 * 'T9' STO 25
'D2' STO 80 D * 'EB' STO
LB9 LB10
»
```

```
LB7
« 2 'J' STO 'T0^2/T2'
EVAL DUP 'T' STO 'T9'
STO 20 'D2' STO 60 D *
'EB' STO LB9 LB10
»

LB8
«
  IF 'D<.3'
  THEN
"do<30 cm, Ber. 3!" HALT
  END 3 'J' STO T0 DUP
'T' STO 'T9' STO 15 'D2'
STO 30 D * 'EB' STO LB9
LB10
»

LB9
«
  IF 'BS<4'
  THEN 5 'D2' STO+
  END
  IF 'D2<EB'
  THEN D2 'EB' STO
  END
  IF 'D<.2'
  THEN
    IF 'T0≤T1'
    THEN 15 'EB' STO
    END
  END
»

LB10
« '7000*I*T*B0/BS' EVAL
'AS' STO 1 'I' STO CLLCD
J "Bereich " LB5 'I'
INCR DROP T0 "%0  : "
LB5 T9 "%  : " LB5 AS
"as-B: " LB5 EB " e-B: "
LB5 3 FREEZE
»

LB11
« " %0 > %3 !" HALT
»
```

▩ steht für: τ

Tastenfolge:

[α] [↱] [cos]

Claus Dachselt
Auf dem Hollen 10
5810 Witten 6

Treffen der Regionalgruppe Rheinland/Ruhrgebiet

Die Regionalgruppe Rheinland/Ruhrgebiet trifft sich auch 1991 jeden zweiten Sonntag im Monat ab 15.00 Uhr.

Treffpunkt:

CCD-Treffen
Witteringstr. 24 (Hinterhof)
4300 Essen 1 (Rüttenscheid)
(Nähe Gruga, Folkwang Museum)

Behandelt werden alle HP-Kleincomputer (HP-19 bis 75) und mittlerweile sehr viel MS-DOS (Hard und Software).

Die genauen Termine für 1991 lauten:

14. April	12. Mai	9. Juni
14. Juli	11. August	8. September
13. Oktober	10. November	8. Dezember

Aufgrund der großen Nachfrage bezüglich des HP-48 wird in Zukunft eine weitere Gruppe mit diesem Rechner als Schwerpunkt aufgebaut. Über Ort und Zeit dieser Treffen werde ich jedoch noch berichten. Es liegen sogar schon Raumangebote seitens der Fachhochschulen in Köln und Bochum vor. Ich wurde in den letzten Wochen von zahllosen Interessenten angerufen, so daß ich eine wirklich große Nachfrage konstatieren kann.

Um einiger Enttäuschung bei Bekanntgabe dieser Termine/Räume vorzubeugen kann ich nur zur Bildung von kleinen Gruppen am Ort aufrufen. Es kann nur einen Treffpunkt geben, Interessenten müssen daher leider eine hoffentlich nicht zu große Fahrzeit in Kauf nehmen. Ich bin jedoch bei der Weitergabe von Anschriften, Interessenten (falls diese dies wünschen) behilflich. Aus den Städten Aachen, Bochum, Dortmund und Köln liegen mir bereits zahlreiche Anfragen vor.

Jochen Haas

Einkommensteuerberechnung

Das Programm ist extra kurz gehalten um jedem die Möglichkeit zu geben seine eigenen Wünsche zu integrieren.

Der Hauptteil der Berechnungen (Lohnsteuertabellen) erfolgt in BST (berechen Steuertabelle). Hier ist der zur Zeit gültige Steuertarif für 1990 ff enthalten. Für diejenigen, denen das Steuerrecht "ein Buch mit sieben Siegeln" ist, folgen hier einige kurze Erklärungen.

Laut Steuergesetzen muß mit einer durch 54 teilbaren Zahl gerechnet werden (desshalb FLOOR), jeder Betrag zwischen den beiden mit 54 als Faktor ermittelten Grenzwerten führt zu dem gleichen Steuerbetrag. Die Variable P ist für die Ermittlung des Steuerbetrages nach Grundtabelle (1) und der Splittabelle (2) und jeweils einen festen Wert.

In BST wird automatisch die Kirchensteuer mit ermittelt. Hier ist der Grundwert mit 9% angenommen worden (in einigen Ländern sind nur 8% Kirchensteuer zu zahlen).

Die Erstattung/Nachzahlung wird von einem Gesamtbetrag aller Steuerzahlungen berechnet, da unterm Strich der gleiche Betrag ermittelt wird und eine getrennte Behandlung deshalb überflüssig erscheint.

Für weitergehende Wünsche sind eventuell noch Spenden an gemeinnützige Vereine bzw. Parteimitgliedsbeiträge zu berücksichtigen. Für diesen Personenkreis: Nach Zeile 23 ... FLOOR P * ... den Betrag der Spende geteilt durch 2 (jedoch maximal 600,-/1200,- DM in Abhängigkeit von Grundtabelle) einsetzen, bzw.direkt von Steuerschuld abziehen!

EST (ermittle Einkommensteuerwerte) fragt den Grundwert ab. Es müssen diese Werte leider wie bisher vorher bestimmt werden und sie gehen nur als Gesamtsumme ein. Dies ändert nichts an den letzlich fälligen Steuerbeträgen, da hinterher sowieso von dem zu versteuernden Einkommen ausgegangen wird. Die unverständlichen Zeilen 1 ... 11 MA ... beruht auf einem Stringverfahren das auch von HP selbst intern im Rechner verwandt wird. Das Prinzip ist es, eine Variable als Programm zu schreiben, die in Listenform alle benötigten Strings enthält und die Listenbehandlung mit erfaßt. Also « Listenklammer links "String1"String2"... "StringN" Listenklammer rechts SWAP GET ». Damit wird jeder String entsprechend der vorgestellten Zahl definiert und entsprechend im Stack zur Verfügung gestellt. Zweckmäßigerweise steht diese Variable dann im HOME, damit alle Programme darauf zugreifen können.

In meinem Fall lautet der String " 'Zeilensprung' 0=Nein;1=Ja". Diese Art der Abfrage erspart mir in vielen Fällen die Flagsetzung für bestimmte Fälle (ist es Euch schon aufgefallen, daß jeder Autor die gleichen Flags verwendet, damit der geplagte USER bei mehreren Unterprogrammen natürlich mit den gleichen Flags unwiderruflich ins Datenchaos gestürtzt wird...).

Bei mir hat diese Abfragetechnik folgenden Hintergrund: Diese Form der Abfrage steuert regelmäßig die Form der Bedingungsabfrage mittels IF-THEN-ELSE Strukturen. Bei "0" wird die ELSE-Bedingung verarbeitet, jeder andere Wert läßt die THEN-Bedingung in Kraft treten.

Für das Programm EST ist aber die Beantwortung mit "0" oder "1" zwingend vorgeschrieben, damit P als Variable den richtigen Wert zugewiesen bekommt (vgl. BST).

Weitere Besonderheiten aus dem Steuerrecht in EST:

Das Programm ist auf einen Arbeitnehmer (AN) mit/ohne Familie ausgelegt. Mehrere Einkommen gehen also als Gesamtsumme ein, ebenso die Steuern, Werbekosten und die Summe der besonderen Vorsorgekosten. Die Pauschalen der allgemeinen Vorsorge sind hier mit dem Satz für AN im öffentlichen Dienst ausgelegt (gekürzter Betrag von max. 3998,- DM; wer also nicht im öffentlichen Dienst ist, muß den Betrag auf 4800,-DM abändern (Zeile 16 und 17)).

Der Betrag der besonderen Vorsorgepauschale beträgt 108,-DM und wird in Abhängigkeit von Grund/ bzw. Splittingtarif behandelt. VS ist hier Ihr ermittelter Betrag der Kosten für die besondere Vorsorge (Kirchensteuer, Lohnsteuerberatungskosten, etc.).

Bei den Vergleichen mit den Pauschalansätzen wird automatisch der jeweil höhere Wert übernommen oder der gesetzlich zulässig Höchstwert berechnet.

Wer eine Einkommensteuer mit mehreren AN berechnen will, muß die Werbekostenpauschale entsprechend multiplizieren (Grundbetrag 2000,-DM), weiterhin wird der Haushaltsfreibetrag für Alleinstehende mit Kindern nicht berücksichtigt (hier ist Grundbetrag 5616,-DM. Dieser wäre anstelle des Kinderfreibetrags von 3024,-DM pro vollangerechnetes Kind anzusetzen (Zeile 28).

Diese Anregungen sollen zeigen, wie man dieses Programm den eigenen Verhältnissen entsprechend umstricken kann. Weiterhin muß bemerkt werden, daß in diesem Programm keine Einkünfte aus Kapitalvermögen bzw. Mieten und Verpachtungen berücksichtigt werden. Wer keine eigenen Ideen hat, kann ja mit mir Kontakt aufnehmen (WARNUNG !!! Lohnsteuerjahresanträge werden nur gegen Honorar bearbeitet !!!).

Für Tüftler noch eine Anregung:

Wie wärs mit einer Formel mit den o. a. Variablen mal in EQ ??? z.B. man nehme

```
48SX EST
     « "SPLITTING?" 11
002  MA + PROMPT 1 + 'P'
     STO "BRUTTO?"
004  PROMPT 'EINK' STO
     "gez STEUER (LST+KS
006  T)?"
     PROMPT 'ST' STO
008  "WERBK?" PROMPT
     'WK' STO
010  "Bes Vorspausch?"
     PROMPT 'VS' STO
012  "KINDER?" PROMPT
     'k' STO EINK DUP 40
014  % → x
       «
016      IF '3998<x'
         THEN 3998
018      ELSE x
         END -
020    »
       IF '2000>WK'
022    THEN 2000
       ELSE WK
024    END -
       IF '108*P<VS'
026    THEN VS
       ELSE 108 P *
028    END - 3024 k * -
BST
030  »  Bytes : 457
     Checksum : # 2A45h

48SX BST
     « P / 54 /.FLOOR 54
002  * → x
       «
004      CASE 5616 x >
           THEN 0
006        END '5617<x'
'x<8153' AND EVAL
008        THEN x ,19 *
1067 -
010        END '8154<x'
'x<120041' AND EVAL
012        THEN x 8100 -
10000 / → y
014          « y 50
151,94 * 1900 y * + *
016  472 +
           »
018        END 'x>120041
' EVAL
020        THEN ,53 x *
22842 -
022        END
         END FLOOR P *
024  DUP DUP 'Einkst'
→TAG SWAP 300 k * -
026  9 * 100 / DUP
         IF 0 <
028      THEN DROP 0
         END 'Kirchenst'
030  →TAG ROT OVER + ST
     - NEG DUP
032      IF 0 <
         THEN NEG
034  'Nachzahlung'
         ELSE
036  'Erstattung'
         END →TAG
038    »
     »  Bytes : 609,5
040  Checksum : # FCEAh

48SX DEL
     « { k VS WK ST EINK
002  P } PURGE
     »  Bytes : 50,5
004  Checksum : # 3794h
```

den gewünschten Erstattungsbetrag und lasse den eingebauten SOLVE-Befehl die Werte der benötigten Werbekosten etc. berechnen.

DEL ist ein Unterprogramm, um vor der Abspeicherung ins Archiv die nicht benötigten Variablen zu löschen und hat weiter keine praktische Bedeutung. Da die Variablen aber nicht gelöscht werden, kann bei einer leichten Veränderung der Grundwerte auch BST allein aufgerufen werden (neuen Betrag zu versteuerndes Einkommen in Stack1 und los gehts ...).

Beispiel für die Berechnung:

Abfragen:	Grund	Splitting
Splitting?	0	1
Brutto?	50000,-	50000,-
gez.Steuer?	4300,-	4300,-
Werbekosten	4800,-	4800,-
bes.Vorsorgepausch.?	580,-	580,-
Anzahl Kinder	3	3
ergibt:		
Steuerbetrag	4036,00	5759,00
Kirchensteuer	282,24	437,31
Nachzahlung	1896,31	18,24

Georg Hoppen

```
485X TREE
     « CLLCD PATH HOME {
002  :0: "HOME" } '$2'
     STO
004    « 1 + 15 TVARS 1
       IF OVER SIZE
006    THEN
         DO GETI DUP 2
008  DISP DUP 5 PICK
     →TAG PATH HOME SWAP
010  $2 SWAP + '$2' STO
     EVAL EVAL 3 PICK $1
012      UNTIL DUP 1
     ==
014      END
       END 3 DROPN
016    » '$1' STO → r
       « 0 $1 HOME $2 {
018  $1 $2 } PURGE r
     EVAL CLLCD DUP SIZE
020  8 * R→B # 83h SWAP
     BLANK SWAP :0: "" +
022  1
       DO GETI OBJ→
024  OBJ→ 9 * R→B 3 PICK
     2 - 8 * R→B 2 →LIST
026  5 ROLL 3 ROLLD SWAP
     DUP →TAG OBJ→ SWAP
028  DROP 2 →GROB REPL 3
     ROLLD
030    UNTIL DUP 1 ==
       END DROP2 PICT
032  STO
       IF 5 FS?
034    THEN PICT RCL
     PR1 DROP
036    ELSE { } PVIEW
       END PICT PURGE
038  »
     »  Bytes : 447,5
040  Checksum : # 9B50h
```

```
HOME
  SYS
  WS
    WSU
      TEST
        WRK
      PLAY
        WRK
      ARCH
        WRK
```

Effektiver Jahreszinssatz

KSK Version (360 Tage/Jahr) - Darlehen, Kredite mit Disagio von Wolfbert Schülle

HP-41 C, 45 Zeilen, 90 Bytes, 13 Regs., SIZE 001, Fix 2.

Mein Programm berechnet den effektiven Jahreszinssatz von Darlehen und Krediten unter Berücksichtigung des Disagios.

Programmablauf:

Start durch XEQ EJZ .

DATUM 1?	Eingabe des Anfangs- / Ausgabedatums R/S (D1)
Datum 2?	Eingabe des End- / Fälligkeits Datums R/S (D2)

(das Format für das Datum ist TT.MM.JJJJ)

%?	Eingabe des Zinnsatzes in % R/S (%)
Kurs?	Eingabe des Ausgabekurses R/S (AKU)
Ausgabe:	Effektiver Jahreszinns (EJZ)

Folgerechnung mit RTN R/S oder R/S R/S

Konventionen:

Datumsformat: TT.MM.JJJJ

Formel:

RW= JJJJ + MM/12 + TT/360

EJZ=%x1000/AKU+100-AKU/D2-D1

Register 00=Rechenwert Datum

LBL A=Umwandlung Datum - Rechenwert

```
01♦LBL "EJZ"
02 "DATE 1?"
03 PROMPT
04 XEQ A
05 STO 00
06 "DATE 2?"
07 PROMPT
08 XEQ A
09 RCL 00
10 -
11 "%?"
12 PROMPT
13 1 E2
14 *
15 "KURS?"
16 PROMPT
17 /
18 X<>Y
19 1 E2
20 LASTX
21 -
22 X<>Y
23 /
24 +
25 RTN
26♦LBL A
27 INT
28 LASTX
29 FRC
30 1 E2
31 *
32 INT
33 LASTX
34 FRC
35 1 E4
36 *
37 X<>Y
38 12
39 /
40 +
41 X<>Y
42 360
43 /
44 +
45 END
```

Tree

Dieses Programm ist von Eric Davis, die Anschrift ist leider nicht bekannt. Die folgenden Kommentare stammen von dem Autor selbst.

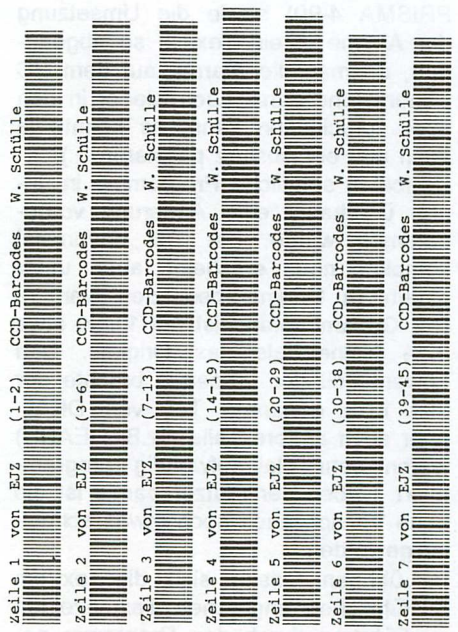
Zeilen

1 - 3:	Pfad zu dem Verzeichnis ermitteln, aus dem das Programm Tree aufgerufen wurde, Listenbeginn festlegen.
4 - 15:	Unterprogramm für die Reihenfolge der Verzeichnisse (später $1)
16:	abspeichern der Liste und des in 1 - 3 ermittelten Weges
19 - 21:	leeres GROB bilden
23 - 32:	Verzeichnisliste nach und nach dem Bild hinzufügen

Achtung: Die Flagabfrage in Zeile 33 - 36 stammt von mir, Flag 5 soll hier das (Nicht-)Vorhandensein des Druckers simulieren und entsprechend soll das Programm die Ausgabe des Bildes regeln. Im Original war lediglich die optische Anzeige vorgesehen (ELSE-Bedingung).

Das nebenstehende Bild zeigt den grafischen Ausdruck.

Georg Hoppen

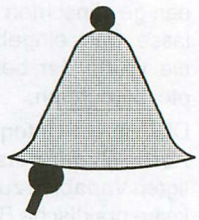


Wolfbert Schülle
Kolpingstraße 5
3570 Stadtallendorf

ALARMVERWALTUNG

auf dem HP48SX

von Günter Schapka



Von HP wird zusammen mit dem Schnittstellenkabel für den HP48SX eine Softwarediskette vertrieben. Diese enthält unter anderem das Programm APPT, mit dem sich Termine über eine Kalenderanzeige gut verwalten lassen.
Leider hat dieses Programm für uns den Nachteil, daß es komplett auf amerikanische Verhältnisse zugeschnitten ist, d.h. Datum im Format MM.DDYYYY. Weiterhin beginnt die Kalenderanzeige mit dem Sonntag als erstem Wochentag, bei uns ist der Montag üblich. Von den Alarmen im Rechner läßt sich ein Textfile zusammenstellen, das zum PC gesandt werden und als Übersicht mit einer Textverarbeitung gedruckt werden kann. - Eine zwar schöne, wie ich meine, aber nicht sehr nützliche Sache.
Als Sicherung für die Alarme kann nur ein kompletter 48-Backup benutzt werden, der wiederum wegen seiner Aufzeichnungsart nicht lesbar ist. Dadurch muß man den kompletten archivierten Rechnerinhalt laden, um an alte Alarme heranzukommen.
Ich habe das Datumformat im gesamten Programm auf die bei uns übliche Schreibweise geändert, weiterhin die Kalenderanzeige über das modifizierte Programm DKAL (von Ralf Pfeifer, PRISMA 3-90) angepaßt, für den Wochentag-Text das Programm DOW eingesetzt (ebenfalls Ralf Pfeifer, PRISMA 4-90) sowie die Umsetzung der Alarme in ein Textfile so abgeändert, da man die Alarme auf dem PC sichern (und damit auch wieder in den 48er zurückladen kann, es könnte ja auch mal ein Absturz passieren ...).
Es folgen sämtliche Programme in denen überhaupt eine Änderung vorgenommen wurde. Ein Teil davon ist komplett neu, erscheint aber unter einem im Original benutzten Namen. Der Gesamtumfang ist mit 10.4K etwa 22% kleiner als das Original. Den Löwenanteil der Ersparnis brachte der hier KAL genannte Teil von DKAL, aber auch andere Teile (z.B. HEADR) waren sehr platzaufwendig programmiert. Neben der Platzersparnis ist die neue Version auch noch etwas schneller geworden.
Im übrigen lassen sich die extrem ausführlichen englischen Textanzeigen zu Führung durch das Programm natürlich noch bedeutend reduzieren, das aber kann jeder nach eigenem Geschmack durchführen.

Folgende Programme entfallen:
DFLIP, MKSTR; GDATES, RAPPTS, GDALRM, M1 bis M7, CADR.
Folgende Programme wurden geändert:
APPTS, TOPC, OVERALL, NOHEAD, GOTO, MHEAD, RC, SETUP2, MOMIN, MOPL, DOKEY, MNTH, LMNTH, MMYY, HEADR, DFRST, MYFLAGS.
Folgende Programme sind neu:
LI→AL, AL→LI, SETU6, DOK6 (alle im Unterverzeichnis APLST), KAL, DOW (beide im Unterverz. MASTR).
Im File MYFLAGS waren folgende Flags gesetzt:
5-12, 35, 40, 41, und 44 (Nummer für die Systemflags #98400000FF0h).
Im geänderten File sind gesetzt:
5-10, 41, 42, 60 und 61 (#18000300000003F0h).

APPTS

```
        « DEPTH
DROPN
          IF FAPPTS
          THEN
DEPTH ROLL APS→MS
PSTMSG
          ELSE
NOAPPTS
          END
        »
MYFLGS {
# 1729385555445154800d
# 0d }
```

LI→AL

```
« "NAME" { { 1 0 }
\Ga } INPUT OBJ→ OBJ→
1 SWAP
  FOR i STOALARM
DROP
  NEXT
»
```

AL→LI

```
« 1 'NA' STO
  DO NA
    IFERR RCLALARM
    THEN 0
    ELSE NA 1 +
    END DUP 'NA'
STO
  UNTIL 0 ==
  END 1 - →LIST
'ALARME' STO 'NA'
PURGE
»
```

SETU6

```
« { "AL→LI" "SEND"
"" "RECV" "LI→AL"
"ABRT" } MENU
»
```

DOK6

```
«
  DO -1 WAIT
    CASE DUP 91.3
==
      THEN OFF
      END DUP 12.1
==
      THEN DROP
'ALARME'
        IFERR CLLCD
SEND
        THEN DROP
"NAME-FILE-I/O Problem"
1 DISP 1 FREEZE
"Check + retry" 2
DISP 2 FREEZE
        ELSE CLLCD
"   OK" 2 DISP 2
FREEZE
        END 12.1
      END DUP 11.1
==
      THEN DROP
AL→LI .1 1500 BEEP
CLLCD
"File: ALARME ok !"
2 DISP 2 FREEZE
11.1
      END DUP 14.1
==
      THEN DROP
RECV CLLCD
"Liste OK !" 2 DISP
2 FREEZE 14.1
      END DUP 15.1
==
      THEN DROP
LI→AL .1 1500 BEEP
CLLCD "ALARME OK !"
2 DISP 2 FREEZE
15.1
      END
    END 1 WAIT
  UNTIL 16.1 ==
  END 16.1
»
```

TOPC

```
« CLLCD
" ALARME ←→ PC" 1
DISP 1 WAIT SETU6 »
```

OVERALL
```
« DROP2
  DO TOPC DOK6
  UNTIL 16.1 ==
  END 'ALARME'
PURGE
»
```

NOHEAD
```
      « DSTR DOW
DSTR TIME TSTR 4 12
SUB + "      " SWAP
+
"   Keine Termine am"
CLLCD 1 DISP 1
FREEZE 3 DISP 3
FREEZE
      »
```

MHEAD
```
      « DSTR DOW
DSTR TIME TSTR 4 12
SUB + "      " SWAP
+ "     Termine f"
220 CHR + "r" +
CLLCD 1 DISP 1
FREEZE 2 DISP 2
FREEZE
      »
```

GOTO
```
    « DROP2 BLKMENU
"Datum (DD.MMYYYY):
dann ENTER."
DSTR →STR -1 2
→LIST INPUT OBJ→
'DSTR' STO
    »
```

RC
```
    « Dy ADR + 7 /
FP 7 * 3 * 6 * 5 +
'COL' STO Dy ADR +
7 / IP 1 + 8 * 1 -
'ROW' STO
    »
```

SETUP2
```
    « DUP 'DSTR'
STO DUP MMYY DUP
DFRST
    »
```

MOMIN
```
    « ROT DROP2
DSTR FP 100 * DUP
IP 1 -
      IF 1 <
      THEN .0001 -
12 +
      ELSE 1 -
      END 100 / 1 +
RDOSCR
    »
```

MOPL
```
    « ROT DROP2
DSTR FP 100 * DUP
IP 1 +
      IF 13 >
      THEN FP
1.0001 +
      ELSE 1 +
      END 100 / 1 +
RDOSCR
    »
```

DOKEY
```
    «
      CASE DUP 11.1
==
        THEN DROP
SRCMAIN REFRESH
11.1
        END DUP
12.1 ==
        THEN DROP
GOTO REFRESH 12.1
        END DUP
36.1 ==
        THEN DYPL
        END DUP
34.1 ==
        THEN DYMIN
        END DUP
35.1 ==
        THEN NWEEK
        END DUP
25.1 ==
        THEN PWEEK
        END DUP
95.1 ==
        THEN MOPL
        END DUP
85.1 ==
        THEN MOMIN
        END DUP
95.2 ==
        THEN YRPLS
        END DUP
85.2 ==
        THEN YRMIN
        END DUP
13.1 ==
        THEN CLEAR
DSTR TIME 100 * IP
100 / "" 0 4 →LIST
BEG 1 CF REFRESH
13.1
        END DUP
14.1 ==
        THEN DROP
OVERALL REFRESH
14.1
        END DUP
15.1 ==
        THEN APPTS
REFRESH 15.1
        END DUP
91.3 ==
        THEN OFF
        END
      END Yr OBJ→
10000 / Mo + 100 /
Dy + 'DSTR' STO
    »
```

MNTH
```
    « DSTR LMNTH
KAL DROP
    »
```

LMNTH
```
    « FP 1 + DUP 32
DATE+ FP 1 + DDAYS
    »
```

MMYY
```
    « DUP IP 'Dy'
STO FP 100 * DUP IP
'Mo' STO FP 10000 *
→STR 'Yr' STO
    »
```

HEADR
```
    « "      "
"Jan. Feb. Mar. Apr.  Mai
Juni Juli Aug. Sep. Okt.
Nov. Dez. "
Mo 5 * DUP 4 - SWAP
SUB + Yr +
    »
```

DFRST
```
    « FP 1 + DOW
LASTARG 2 / 2 - DUP
'ADR' STO 3 DROPN
    »
```

DOW
```
    « 0 TSTR 1 2
SUB
"MoDiMiDoFrSaSo"
"MOTUWETHFRSASU"
ROT POS DUP 1 + SUB
    »
```

KAL
```
    « LASTARG ROT 3
FREEZE 3 * " Mo   "
HEADR + "      " 1
1.01199 7 PICK
DDAYS 148729 + 7
MOD SUB DUP DUP + +
+
"  1  2  3  4  5  6  7  8
9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26
27 28 29 30 31"
1 4 ROLL SUB + 1 7
      FOR n 1 21
SUB LASTARG + OVER
SIZE SUB SWAP n
DISP
      NEXT DROP2
    »
END
```

Günter Schapka
Rebusgasse 11
D6100 Darmstadt

Programmaufruf aus anderen Verzeichnissen

"PATH" beim HP28/48

Wer hat sich nicht schon einmal über die fehlende PATH-Anweisung beim HP28/48 geärgert, die es ermöglichen würde, nach Programmen bzw. Objekten in anderen Verzeichnissen suchen zu lassen.
In der Welt der Personal- bzw. Homecomputer ist diese Anweisung an das Betriebssystem eine unentbehrliche, ermöglicht sie doch den bequemen Aufruf von Dienstprogrammen sowie bei DOS-Rechnern den Aufruf von Betriebssystemutilities bzw. Dienstprogrammen im \DOS\-Verzeichnis, da das Betriebssystem bekanntlich nur das Allernötigste enthält, damit der ohnehin knappe Speicher der IBM-Kompatiblen nicht noch weiter gekürzt wird.
Befinde ich mich bei meinem HP28/48 in einem Verzeichnis {HOME PRISMA ANWEND} und will ein Programm im Verzeichnis {HOME EIGEN UTILS} aufrufen, so muß ich erst mit viel Tipparbeit in dieses überwechseln.
Ralf Schramm hat sich hier einen Trick überlegt, der diesem Problem auf den Leib rücken könnte:
Er hat ein kleines Programm geschrieben, das sich als temporäres Menü in das System einklinkt. Die Schwierigkeit bestand darin, die zugehörigen Programme zu finden, da die Unterprogramme keinen Platz mehr in diesem Menü gefunden hätten. Ein solches Menü hat dann folgenden Aufbau:

```
{{"PRG 1" «PRG1»}{"PRG 2"«PRG2»}}
```

Man sieht nun, von links nach rechts, die beiden Tastentexte "PRG 1" und "PRG 2". Drückt man nun eine der beiden Softkeys, so wird das zugehörige Programm ausgeführt. Dies geht aber nur, wenn man sich im richtigen Menü befindet.
Evaluiert man beim HP28/48 einen Objektnamen (EVAL), so sucht der Rechner zuerst im aktuellen Verzeichnis und danach im HOME-Verzeichnis nach diesem Objekt. Was liegt also näher, als im HOME-Verzeichnis ein Parameterobjekt zu speichern, das die Pfadnamen der Programme enthält:

```
{HOME LIBRARY} 'PPAR' STO
```

Außerdem benötigt man noch ein Programm 'RUN', das die Pfadumschaltung im HOME-Verzeichnis durchführt:

```
«PATH →p
  «EVAL p EVAL»
 »'RUN' STO
```

Nun kann man die Menüstruktur folgendermaßen umschreiben:

```
{{"PRG 1" «PPAR PRG1 + RUN»} {"PRG
2" «PPAR PRG2 + RUN»}}
```

Damit sollten nun Programme von allen Pfaden aus gefunden werden können. Das Parameterobjekt kann man ja von einem Installationsprogramm erstellen lassen:

```
«PATH HOME VARS SWAP
  «PATH →p
    «EVAL p EVAL»
  »'RUN' STO
'PPAR' STO ORDER»
```

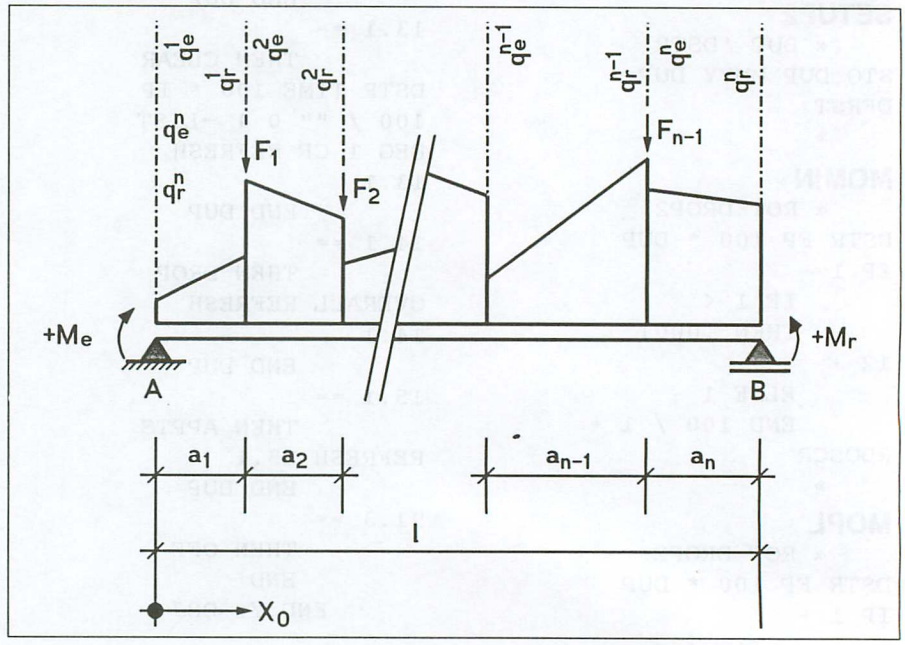
Ralf Schramm
Frauenlobstraße 3
6500 Mainz
Überarbeitung: mm

Schnittgrößen für Einfeldträger

unter beliebiger Belastung

Das vorliegende Programm "EINF" ermittelt für einen Einfeldträger mit beliebiger Belastung und vorgegebenen Randmomenten die Auflagerreaktionen sowie Ort und Größe des maximalen Feldmomentes.
Als Belastung werden je Abschnitt die linke und die rechte Ordinate der Streckenlast in diesem Bereich, ql und qr, sowie, falls vorhanden, eine Einzellast F am rechten Abschnittende eingegeben. Nach Vorgabe der Randmomente gibt das Programm die Auflagerkräfte A und B aus; es können an dieser Stelle beliebig oft über den Programmteil "MOM" neue Randmomente eingegeben werden, um zugehörige Auflagerreaktionen zu ermitteln.
Durch das Starten des Programms "MF" werden im Anschluß an die zuletzt vorgegebenen Randmomente Ort und Größe des zugehörigen maximalen Feldmomentes ermittelt und angezeigt. Zusätzlich erfolgt zur Kontrolle die Anzeige der Gesamtlänge l des Trägers.
Vorzeichenkonvention: Positive Rand- und Feldmomente erzeugen Zug auf der Unterseite des Trägers !
Hinweis: Nach der Berechnung des maximalen Feldmomentes über "MF" muß das Programm zur Berechnung neuer Auflagerkräfte wieder über "EINF" gestartet werden; der Aufruf von "MOM" führt sonst zu falschen Ergebnissen.
(Beispiel siehe nächste Seite)

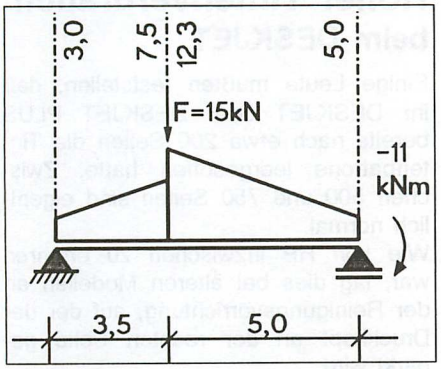
Claus Dachselt
Auf dem Hollen 10
D5810 Witten 6



```
EINF
« "Anzahl Abschnitte" {
":n:" { 1 0 } } INPUT
OBJ→ 'N' STO { 1 N } 0
CON DUP DUP2 'F' STO 'A'
STO 'QL' STO 'QR' STO 0
DUP DUP2 OVER 'R' STO
'U' STO 'C' STO 'T' STO
'L' STO 1 N
  FOR j j 'K' STO BER
":a:" INPUT OBJ→ DUP 'L'
STO+ 'A' AUFB BER {
":ql:
:qr:
:F:" { 1 0 }
} INPUT OBJ→ 'F' AUFB
'QR' AUFB 'QL' AUFB
  NEXT 1 'J' STO LE1
»

BER
« "Bereich " K →STR +
»

AUFB
« DUP EVAL 3 ROLL K SWAP
PUT SWAP STO·
»
```

```
LE1
« LE2 'R' STO+ Q1 Q2 +
'S' STO '-(Q1+Q2/2)*AL^2
/S+R*S*AL/2' EVAL 'U'
STO+ 'F' AUS DUP 'S*AL/2
' EVAL + 'T' STO+ R *
'U' STO+
  IF 'L==R'
  THEN MOM
  ELSE 'J' INCR DROP RET
  END
»

LE2
« 'QL' AUS 'Q1' STO 'QR'
AUS 'Q2' STO 'A' AUS DUP
'AL' STO
»

AUS
« EVAL J GET
»

MOM
« "Endmomente" {
":M-l:
:M-r:" { 1 0 } }
INPUT OBJ→ SWAP DUP 'M'
STO - NEG U + R / DUP
'VB' STO T - NEG 'VA'
STO ANZ 3 FREEZE
»
```

```
RET
« LE1
»

ANZ
« 1 'I' STO CLLCD VA
"V-A: " LB5 VB "V-B: "
LB5
»

LB5
« SWAP + I DISP 'I' INCR
DROP
»

MF
« 1 CF 0 'U' STO VA 'V'
STO
  IF 'V<0'
  THEN 1 SF
  END 1 'J' STO LE3
»

LE3
«
  IF 'L==U'
  THEN LE8
  ELSE LE2 'U' STO+ 'V'
'(Q1+Q2)*AL/2' EVAL STO-

'((Q1/2+Q2)*AL/3+V)*AL'
EVAL 'M' STO+ V 'W' STO
    IF 1 FS?
    THEN V NEG 'W' STO
    END
    CASE 'W>0'
      THEN LE4
      END LE5
    END
  END
»

LE4
« 'V' 'F' AUS STO- 'J'
INCR DROP V 'W' STO
  IF 1 FS?
  THEN V NEG 'W' STO
  END
  IF 'W>0'
  THEN LE3
  ELSE LE8
  END
»

LE5
« '(Q2-Q1)/AL' EVAL 'S'
STO
  IF 'S≠0'
  THEN LE6
  ELSE V Q2 / 'Y' STO
LE7
  END
»

LE6
« '√(Q2^2+2*V*S)' EVAL
'Y' STO
  IF 1 FS?
  THEN Y NEG 'Y' STO
  END '(Y-Q2)/S' EVAL
'Y' STO LE7
»

LE7
« Y DUP 'U' STO+ 'V' STO

'(V*S/3+Q2/2)*V^2' EVAL
'M' STO+ LE8
»

LE8
« ANZ 'I' INCR DROP L
"1   : " LB5 'I' INCR
DROP U "xo : " LB5 M
"Mf : " LB5 3 FREEZE
»
```



Im folgenden bedeuten:

▼ = Cursor-Down Taste

↵ = ENTER Taste

Zum Aufruf des Variablenmenüs Taste **VAR** drücken.

(schwarze Tasten mit invertierter Schrift sind Softkeys)

Anzeige	*Eingabe / Tastenfolge*		
	EINF		
Anzahl Abschnitte			
n:		2	↵
Bereich 1			
a:		3.5	↵
Bereich 1			
ql:			
qr:			
F:	3	▼ 7.5	▼
		15	↵
Bereich 2			
a:		5	↵
Bereich 2			
ql:			
qr:			
F:	12.3	▼ 5	▼
		0	↵
Endmomente			
M-l:			
M-r:	0	▼ -11	↵
V-A: 36.091			
V-B: 40.534			
V-A: 36.091			
V-B: 40.534			
l: 8.500			
Xo: 3.724			
Mf: 99.057			

Fortsetzung mit **MF**

< Infos Infos Infos Infos Infos Infos Infos >

Assembler-ROM
für den HP48SX

Für den HP48SX gibt es von W&W ein Assembler/Disassembler-ROM. Es befindet sich gerade im Test, sodaß wir im nächsten PRISMA einen Bericht für all diejenigen zu präsentieren hoffen, die schon länger einmal eine Entdeckungsreise in ihr neues Wunderding unternehmen wollten.
Die Steckkarte paßt in einen der beiden Steckplätze des Rechners, die Dokumentation enthält aber leider keinerlei Angaben über die Innereien des HP48, hier wird auf die Spezifikation des HP71 verwiesen, der den gleichen bzw. ähnlichen Prozessor enthält.
Die Adressaufteilung des HP48 kann man sich ja im PRISMA 5-6/90 auf Seite 37 ansehen. Hier hatte ich bereits versucht einen kleinen Überblick zu geben, soweit mir Informationen vorlagen.
Was natürlich fehlt ist die genaue Funktionsbeschreibung z.B. des Display-Controllers, um direkt und schnell an den Displayinhalt zu gelangen.

mm

HP48 NEWS

Hewlett Packard will noch dieses Jahr ein HP41CV Emulator Modul auf den Markt bringen, das die Ausführung von Programmen ermöglicht, die ohne die Anwendung von Funktionen von Einsteckmodulen programmiert worden sind.

Das wäre in etwa dasselbe wie das gleiche Modul für den HP71.

Von W&W soll es ebenfalls noch dieses Jahr einiges Zubehör rund um den HP48 geben wie etwa eine HP-IL Anbindung oder eine Anschlußmöglichkeit für einen Barcodeleser.

Ebenso sind Module für die Gebiete Vermessung, Navigation, Baustatik, Statistik und Mathematik geplant. Das angekündigte Assembler/Disassembler Modul ist ja bereits Wirklichkeit geworden.

Im Zusammenhang mit dem **HP48** ist inzwischen so einiges an **Literatur** erschienen, die ich kurz aufzählen möchte:

HP48 Programmer's Reference Manual
Hewlett-Packard
Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, USA
Recorder Number: 00048-90054

Dieses Buch enthält im wesentlichen ein Verzeichnis der Befehle des HP48SX. Außerdem beinhaltet es Tabellen über Fehler- und Statusmeldungen, die Einheiten sowie die Systemflags. Es ist sozusagen die kleine Taschenbibel des ambitionierten HP48 Anwenders.

The HP48SX Handbook
von James Donelly
Armstrong Publishing Company,
3135 NW Ashwood Drive
Corvallis, OR 97330, USA

Dieses Buch ist ein Referenzhandbuch für die Gleichungslöserbibliothek des HP48SX. Es behandelt Objekte, Grafiken, Systemoperationen, Speicher, Statistik, Programmierung und mehr.

HP41 / HP48 Transitions
William C. Wickes
Larken Publications,
4517 NW Queens Avenue,
Corvallis, OR 97330, USA
ISBN 0-9625258-2-0

In diesem Buch wird die logische Weiterentwicklung des HP41 zum HP48 beschrieben. In diesem Buch findet man, wie Prozeduren des HP41 auf den HP48 umgschrieben werden können, ebenso ob Methoden der HP41 Programmierung weiter verwendet werden können bzw. umgeändert werden müssen.
Es behandelt auch viele Funktionen des HP41CX.

HP48 Graphics
Grapivine Publications
PO Box 118,
Corvallis, OR 97330, USA

Dieses Buch beschäftigt sich, wie sein Name schon sagt, mit der Anwendung von grafischen Objekten beim HP48, damit der Anwender das ganze Potential dieser Funktionen nutzen kann.

Kermit, A File Transfer Protocol
Frank da Cruz
Educalc Mail Store
27953 Cabot Road
Laguna Niguel, CA 92677, USA
ISBN 0-932376-88-6

Dieses Buch beschäftigt sich mit der Geschichte des Datenübertragungsprotokolls KERMIT, das im HP48 für die Datenübertragung benutzt wird. Es enthält prinzipielle Erläuterungen sowie detaillierte Anleitungen zur Bedienung. Datenübertragungsprogramme, die mit dem KERMIT-Protokoll arbeiten, existieren so gut wie für jeden Rechner dieser Erde, das ist keine Übertreibung!

mm

Umtausch
DESKJET 500 Handbücher

Eine Serie der DESKJET 500 Handbücher scheint teilweise falsch gebunden worden zu sein. Dabei wurden ganze Kapitel vertauscht.
Diese Handbücher werden nach HP-Angaben kostenlos umgetauscht. Die Umtauschprozedur läuft wie folgt:
Man soll das Handbuch dort, wo man den Drucker bezogen hat, reklamieren, indem man es dorthin zurückbringt bzw. schickt. Diese Reklamationen gehen dann weiter an einen Hr. Jahn in Böblingen, der ein korrekt gebundenes Ersatzhandbuch zurückschickt.

Hoher Tintenverbrauch
beim DESKJET

Einige Leute mußten feststellen, daß ihr DESKJET oder DESKJET PLUS bereits nach etwa 200 Seiten die Tintenpatrone leergesoffen hatte. Zwischen 500 und 750 Seiten sind eigentlich normal.
Wie von HP inzwischen zu erfahren war, lag dies bei älteren Modellen an der Reinigungsvorrichtung, auf der der Druckkopf an der rechten Seite geparkt wird.

Betroffen sind Geräte mit den Seriennummern *kleiner* 2937...

Damit können sich alle Besitzer des DESKJET 500 als nicht betroffen fühlen; Leute mit einem DESKWRITER, das ist die Ausführung mit der Hochgeschwindigkeitsschnittstelle für den APPLE Macintosh, können ebenfalls betroffen sein.
Man kann an der Farbe des Plastikschlittens, auf dem sich die Gummidichtung für den Druckkopf in seiner Parkposition befindet, erkennen, ob man eine alte oder neue Reinigungsvorrichtung eingebaut hat:
Die alte Vorrichtung ist dunkelgrau oder schwarz, die neue ist beige oder weiß.

Abhilfe:
Man kann von HP ein kostenloses Upgrade Kit erhalten, das die Nummer P/N 02276-60106 trägt.
Jedem Druckkopf, der jetzt verkauft wird, sollte diese Information beiliegen. HP bittet nur darum, bei der Beschaffung des Upgrade-Kits unbedingt die Seriennummer des Gerätes mit anzugeben, damit man weiß, wie lange die Austaschaktion noch andauern muß.

mm

HP28S SYSEVAL-Adressen

Im PRISMA 2/90 war von Peter Röhl die grundsätzliche Verwendung der Funktion SYSEVAL in Zusammenhang mit der Maschinenspracheprogrammierung erläutert worden. Er hatte für eine Reihe spezieller Möglichkeiten einige Einsprungadressen angegeben.
Die Funktion SYSEVAL ist eigentlich nichts anderes als eine *GOSUB Adresse*, d.h. ein Sprung zu einem an *Adresse* anfangendem Unterprogramm, das daraufhin ausgeführt wird.
Nachfolgend möchte ich eine mehr oder weniger komplette Liste aller bekannten Einsprungadressen der Betriebssystemversion 2BB liefern, die von dem Holländer Eric Toonen ermittelt wurde.
Wichtig dabei ist, daß man diese Adressen nur in Zusammenhang mit der eben genannten Version des HP28**S** sehen kann, bei den alten Betriebssystemversionen 1BB und 1CC des HP28**C** stimmen diese Adressen nicht; eine Benutzung kann zum Absturz führen !!
Wie weiß ich nun, welches Betriebssystem in meinem Rechner ist ?
#Ah SYSEVAL <ENTER> eintippen, es erscheint die Copyrightmeldung mit der Betriebssystemversion im Display.

Die folgende Liste ist mit Sicherheit nicht fehlerfrei und auch nicht von Hewlett Packard abgesegnet, es kann also keine Garantie für die Richtigkeit der Adressen übernommen werden.
Entdeckt jemand Fehler, so möge er diese bitte auch dem Author zukommen lassen.

Aufbau der Liste:

0000A ! Bezeichnung
| | | |
| | | Dies ist der **Name der Funktion**, der sich hoffentlich selbst erklärt.
| | In dieser Spalte ist markiert, daß diese Funktion zum Absturz führt. D.h. niemals diese Einsprungadresse bei einer SYSEVAL Anweisung benutzen:
| | a) Der Rechner stürzt ab, wenn man diese Adresse aus dem RAM heraus aufruft oder
| | b) dieses Unterprogramm kann nicht im Stack-Modus aufgerufen werden, sondern nur von speziellen Funktionen wie FORM, UNITS oder CATALOG oder
| | c) die Funktion legt ein Objekt im Stack ab, das von den vorhandenen Displayroutinen nicht verarbeitet werden kann und somit einen Absturz verursacht oder
| | d) diese Funktion ist Teil einer Struktur wie "IF" "THEN" "ELSE". Diese können nur in einem internen Format bearbeitet werden.

Dies ist die **hexadezimale Adresse**:
Entweder man ist im Modus HEX (Menü Binary, Softkey HEX ist mit einem kleinen Quadrat gekennzeichnet) oder man gibt die Zahl mit einem kleingeschriebenen "h" am Ende ein, sie wird dann automatisch umgewandelt.
z.B. #Ah oder #A, wenn der HEX-Modus gesetzt ist.

Folgende Objekttypen wurden in der Liste verwendet:

code	Programmcode Anfang (Nur für internen Gebrauch des Rechners !)
pt	Zeiger (pointer) mit 20 Bit Länge (kann über Tastatur nicht eingegeben werden!)
real	Real-Zahl
double	doppelt genaue Real-Zahl (kann über Tastatur nicht eingegeben werden !)
complex	(... , ...) Komplexe Zahl
doublecomplex	doppelt genaue Komplexe Zahl (kann über Tastatur nicht eingegeben werden !)
byte	Byte (8 Bit) (kann über Tastatur nicht eingegeben werden !)
array	[..] Matritze (Array)
string	".." Zeichenkette
binary	#.. Binärzahl
list	{..} Liste
algebraic	'..' algebraisches Objekt
forth	Start einer Forth-Routine (Nur für internen Gebrauch des Rechners !)
name	'..' jede Zeichenkette, die einen Namen darstellen darf
globalname	'..' Namen einer globalen Variable
localname	'..' Namen einer lokalen Variable
continue	Ende einer Forth-Routine (Nur für internen Gebrauch des Rechners !) (kann über Tastatur nicht eingegeben werden !)
boolean	Wahr/Falsch (kann über Tastatur nicht eingegeben werden !)
o029E1	unbekannte Bedeutung
o02A2C	unbekannte Bedeutung
o02AB8	unbekannte Bedeutung
o02C96	unbekannte Bedeutung
o02D5C	unbekannte Bedeutung

Mit besonderer Vorsicht sind folgende Funktionen zu behandeln:
Namen wie *func_obj* benötigen ein Objekt des Typs 'obj' im Stack 1:.
Namen wie *func_1obj2obj* benötigen je ein Objekt des Typs 'obj' im Stack 1: und 2:.
Namen wie *obj_value* geben ein Objekt in den Stack zurück !
Namen wie *obj_func_obj* und *obj_func_1obj2obj3obj* sollten nach den letzten Erklärungen nun klar sein, was sie benötigen: Es werden 3 Objekte im Stack erwartet und eines wieder zurückgegeben.
Sollten eine der Voraussetzungen für den Aufruf einer Funktion fehlen, so kann dies zum Absturz des Rechners führen !!
Nur komplett großgeschriebene Funktionen überprüfen vor ihrer Ausführung den korrekten Zustand des Rechners wie den Syntax der Objekte in den Stackebenen bzw. ob überhaupt die erforderliche Stacktiefe vorhanden ist.

0000A		version
003B4		selftest_nonrepeating
003C1		selftest_repeating
003CE		keyboard_test
003DB		reset&off
00F09		version_2BB
01011		display_on
01023		display_off
0109F		batteries_low?
0112A		clear_lcd
011CA		clock
011D4		pt_12
0200E		pt_depth
0204A		dup
0206E		dup2
0209B		dupn_pt
020E5		swap
02106		drop
0211A		drop2
02130		dropn_pt
02157		rot
02184		over
021A4		pick_pt
021E7		roll_pt
02231		rolld_pt
023D4		pt_elements_array
028FC	!	code
02911	!	pt
02933	!	real
02955	!	double
02977	!	complex
0299D	!	doublecomplex
029BF	!	byte
029E1	!	o029E1
02A0A	!	array
02A2C	!	o02A2C
02A4E	!	string
02A70	!	binary
02A96	!	list
02AB8	!	o02AB8
02ADA	!	algebraic
02C67	!	forth
02C96	!	o02C96
02D12	!	globalname
02D37	!	localname

02D5C	!	o02D5C
02F90	!	continue
030F3		boolean_KEY?_&ifso_scancode
035E6	!	array_errors0xx
036BD		error_pt
03713		ERRM_pt
03829		LAST_do
03856		LAST_force
03B5E		pt_-1
03B82		+_1string2string
03C3F		+_1list2list
03C3F		+_1program2program
03D75		+_1byte2string
03D81		+_1any2list
03EEB		→program_pt
03EFF		→list_pt
03F13		→algebraic_pt
0407F		string_""
04089		list_{ }
04093	!	algebraic_'
041CF		SUB_1pt2pt3string
043AD		binary→pt
04567		name→string
045E7		R→C_1real2real
04700		C→R_complex
048A4		garbage_collect
048C1		pt_mem
05410		CONJ_real
05410		RE_real
054E8		EVAL_anyBUTalgebraic
05627		hop
05EEF		STO_localname
05FB3		executable_1pt_code2pt_2
06095		executable→name
06B87		pt_2047
06C65		NOT_boolean
06CB9		AND_1boolean2boolean
06CE8		OR_1boolean2boolean
06E8A		boolean_==_1pt2pt
070EB	!	pt_real
070F5	!	pt_complex
070FF	!	pt_array
07109	!	pt_string
07113	!	pt_list
0711D	!	pt_name
07127	!	pt_local
0712C	!	pt_pt
07131	!	pt_forth
0713B	!	pt_algebraic
07145	!	pt_o02AB8
0714F	!	pt_binary
07159	!	pt_byte
07163	!	pt_local
0716D	!	pt_o02D5C
07177	!	pt_double
07181	!	pt_doublecomplex
0718B	!	pt_o02A2C
07195	!	pt_o02C96
0719F	!	pt_o029E1
071A9		pt_0
071B3		pt_1
071BD		pt_2
071C7		pt_3
071D1		pt_4
071DB		pt_5
071E5		pt_6
071EF		pt_7
071F9		pt_8
07203		pt_9
0720D		pt_10
07217		pt_11
07221		pt_12
0722B		pt_13
07235		pt_14
0723F		pt_15
07249		pt_16
07253		pt_17
0725D		pt_18
07267		pt_19
07271		pt_20
0727B		pt_21
07285		pt_22
0728F		pt_23
07299		pt_24
072A3		pt_25
072AD		pt_26
072B7		pt_27
072C1		pt_28
072CB		pt_29
072D5		pt_30
072DF		pt_31
072E9		pt_32
072F3		pt_33
072FD		pt_34
07307		pt_35
07311		pt_36
0731B		pt_37
07325		pt_38
0732F		pt_39
07339		pt_40
07343		pt_41
0734D		pt_42
07357		pt_43
075DC		DUP
075F6		DUP2
07610		SWAP
0762A		DROP
07644		DROP2
0765E		ROT
07678		OVER
07692		DEPTH
076B1		DROPN
076CB		DUPN
076E5		PICK
076FF		ROLL
07719		ROLLD
07733		CLEAR
07752		→LIST
0776C		R→C
077A4		RE
077F0		IM
07832		SUB
0785A		SUB_1real2real3string
0786E		SUB_1real2real3list
0786E		SUB_1real2real3program
07888		LIST→
078A6		algebraic→_algebraic
078A6		LIST→_list
078A6		program→_program
078C0		C→R
078EE		SIZE
0792A		SIZE_string
0793E		SIZE_list
07952		SIZE_array
0796C		POS
07994		POS_1any2list
07994		POS_1any2program
079AE		→STR
079C8		STR→
079EC		NUM
07A10		CHR
07A34		TYPE
07AE4		→ARRY
07B0C		→ARRY_real
07B25		→ARRY_list
07B85		ARRY→
07BA3		ARRY→_array
07BD6		RDM
07BFE		RDM_1list2array
07C21		RDM_1list2name
07C5E		CON
07CAE		CON_1complex2list
07CAE		CON_1real2list
07CDB		CON_1complex2array
07CF9		CON_1real2name
07D1C		CON_1complex2name
07D4A		IDN
07D7C		IDN_real
07DB8		IDN_name
07DE1		TRN
07E09		TRN_name
07E32		PUT
07E96		PUT_1any2list3name
07EAA		PUT_1any2real3name
07F04		PUT_1complex2list3array
07F04		PUT_1real2list3array
07F18		PUT_1complex2real3array
07F18		PUT_1real2real3array
07F63		PUT_1any2list3list
07F77		PUT_1any2real3list
07FB9		PUTI
0801D		PUTI_1any2list3name
08031		PUTI_1any2real3name
0808B		PUTI_1complex2list3array
0808B		PUTI_1real2list3array
0809F		PUTI_1complex2real3array
0809F		PUTI_1real2real3array
080EA		PUTI_1any2list3list
080FE		PUTI_1any2real3list
0814A		GET
081AE		GET_1list2name
081C2		GET_1real2name
08208		GET_1list2array
0821C		GET_1real2array
0823F		GET_1list2list
08253		GET_1real2list
08277		GETI
082DB		GETI_1list2name
082EF		GETI_1real2name
08335		GETI_1list2array
08349		GETI_1real2array
0838F		GETI_1list2list
083A3		GETI_1real2list
083EE		pt_46
087C3		SAME
087EC		AND
0883C		AND_1real2real
0886F		OR
088BF		OR_1real2real
088F2		NOT
0892E		NOT_real
08952		XOR
089A2		XOR_1real2real
089D5		==
08A2F		==_1any2any
08A53		≠
08AAD		≠_1any2any
08AD6		<
08B26		<_1real2real
08B4A		>
08B9A		>_1real2real
08BBE		≤
08C0E		≤_1real2real
08C32		≥
08C82		≥_1real2real
08CA6		=
08D24		NEG
08D70		ABS
08DBC		CONJ
08E08		π
08E36		MAXR
08E64		MINR
08E92		e
08EC0		i
08EEE		+
08FAC		+_1list2any
08FD0		-
09076		*
09144		/
091FE		^
0926C		^_1real2real
0928F		pt_772
092DB		INV
09327		ARG
09359		ARG_real
0937D		P→R
093B9		P→R_array
09469		R→P
094A5		R→P_real
094B9		R→P_array
094E2		SIGN
09524		√X
09556		√X real
0958E		X^2
095CA		X^2_real
095DE		X^2_complex
09602		SIN
09644		COS
09686		TAN
096C8		SINH
0970A		COSH
0974C		TANH
0978E		ASIN
097C0		ASIN_real
09802		ACOS
09834		ACOS_real
09876		ATAN
098B8		ASINH
098FA		ACOSH
0992C		ACOSH_real
09969		ATANH
0999B		ATANH_real
099DD		EXP
09A1F		LN
09A51		LN_real
09A89		LOG
09ABB		LOG_real
09AF3		ALOG
09B35		LNP1
09B6D		EXPM
09BA5		FACT
09BDD		IP
09C15		FP
09C4D		FLOOR
09C85		CEIL
09CBD		XPON

```
09CF5   MAX
09D41   MIN
09D8D   MOD
09DD9   MANT
09E11   D→R
09E49   R→D
09E77   →HMS
09E9B   HMS→
09EBF   HMS+
09EE3   HMS-
09F07   CNRM
09F2B   RNRM
09F4F   DET
09F73   DOT
09F97   CROSS
09FBB   RSD
09FE9   %
0A035   %T
0A081   %CH
0A0C3   RAND
0A0DD   RDZ
0A101   COMB
0A125   PERM
0A149   LCD→
0A163   →LCD
0A187   DGTIZ
0A1A1   MENU
0A1CF   RCL
0A1F7   RCL_name
0A220   STO
0A252   STO_globalname
0A26C   PURGE
0A294   PURGE_globalname
0A2A8   PURGE_list
0A32B   MEM
0A359   ORDER
0A377   ORDER_list
0A396   CLUSR
0A3B0   KILL
0A3CA   ABORT
0A3E4   ERRN
0A3FE   ERRM
0A418   EVAL
0A450   IFTE
0A4C8   IFTE_1any2any3real
0A500   IFT
0A51E   IFT_1any2any
0A54C   SYSEVAL
0A56A   SYSEVAL_binary
0A5A6   DISP
0A5D4   RND
0A616   BEEP
0A63A   →NUM
0A654   LAST
0A672   pt_517
0A73A   WAIT
0A75E   CLLCD
0A778   KEY
0A792   CLMF
0A7B6   SF
0A7D4   SF_real
0A7F3   CF
0A811   CF_real
0A830   FS?
0A84E   FS?_real
0A86D   FC?
0A88B   FC?_real
0A8AF   DEG
0A8D3   RAD
0A8F7   FIX
0A915   FIX_real
0A92F   SCI
0A94D   SCI_real
0A967   ENG
0A985   ENG_real
0A99F   STD
0A9B9   FS?C
0A9D7   FS?C_real
0AA0A   FC?C
0AA28   FC?C_real
0AA60   BIN
0AA7A   DEC
0AA94   HEX
0AAAE   OCT
0AAC8   STWS
0AAEC   RCWS
0AB06   RCLF
0AB20   STOF
0AB3E   RCLF_do
0AB68   STOF_binary
0ABBC   STOΣ
0ABD6   CLΣ
0ABF0   RCLΣ
0AC0A   Σ+
0AC38   Σ-
0AC52   NΣ
0AC6C   CORR
0AC86   COV
0ACA0   MAXΣ
0ACBA   MEAN
0ACD4   MINΣ
0ACEE   SDEV
0AD08   TOT
0AD22   VAR
0AD3C   LR
0AD56   PREDV
0AD7A   UTPC
0AD9E   UTPN
0ADC2   UTPF
0ADF6   UTPT
0AE0A   SCLΣ
0AE24   DRWΣ
0AE3E   COLΣ
0AE62   SINV
0AE80   SINV_name
0AEF9   SNEG
0AF17   SNEG_name
0AF90   SCONJ
0AFAE   SCONJ_name
0B013   STO+
0B063   STO+_1name2complex
0B063   STO+_1name2real
0B090   STO+_1complex2name
0B090   STO+_1real2name
0B0CC   STO+_1name2array
0B0EF   STO+_1array2name
0B118   STO-
0B168   STO-_1name2complex
0B168   STO-_1name2real
0B195   STO-_1complex2name
0B195   STO-_1real2name
0B1D6   STO-_1name2array
0B1F9   STO-_1array2name
0B222   STO/
0B272   STO/_1name2complex
0B272   STO/_1name2real
0B29F   STO/_1complex2name
0B29F   STO/_1real2name
0B303   STO/_1name2array
0B326   STO/_1array2name
0B34F   STO*
0B39F   STO*_1name2complex
0B39F   STO*_1name2real
0B3D6   STO*_1complex2name
0B3D6   STO*_1real2name
0B435   STO*_1name2array
0B485   STO*_1array2name
0B5C6   EXGET
0B5EA   EXSUB
0B622   OBSUB
0B646   OBGET
0B66A   FORM
0B6A2   COLCT
0B6DA   EXPAN
0B712   ISOL
0B736   QUAD
0B75A   SHOW
0B788   TAYLR
0B7B6   d/dx
0B806   d/dx_1algebraic2complex
0B806   d/dx_1algebraic2real
0B815   d/dx_1name2complex
0B815   d/dx_1name2real
0B82E   d/dx_1algebraic2algebraic
0B84D   RCEQ
0B871   STEQ
0B88B   ROOT
0B8E1   ∫
0B91D   ∫ 1real2list3algebraic
0B91D   ∫ 1real2list3program
0B91D   ∫ 1real2list3real
0BB53   ASR
0BB77   RL
0BB9B   RLB
0BBBF   RR
0BBE3   RRB
0BC07   SL
0BC2B   SLB
0BC4F   SR
0BC73   SRB
0BC97   R→B
0BCBB   B→R
0BCDF   CONVERT
0BD21   INDEP
0BD45   PMIN
0BD69   PMAX
0BD8D   AXES
0BDB1   CENTR
0BDD5   RES
0BDF9   *H
0BE1D   *W
0BE41   DRAW
0BE5B   PIXEL
0BE7F   DRAX
0BE99   PR1
0BEB3   PRST
0BECD   PRSTC
0BEE7   CR
0BF01   PRUSR
0BF1B   PRVAR
0BF43   PRVAR_list
0BF5D   PRMD
0BF77   PRLCD
0BF91   CRDIR
0BFB5   PATH
0BFCF   HOME
0BFE9   VARS
0BFFD   DUP&RCL_name
0C1DD   two_pt_1real2real
0C1FB   two_real_1pt2pt
0C25F   LAST_off
0C291   !  array_errors2xx
0C36A   !  need_0
0C3BD   !  need_3
0C3CD   !  need_2
0C3EA   !  need_1
0C484   !  need_n&pt
0C53B      real→pt
0C610      pt→real
0C784      EVAL_algebraic
0C79C   !  pass_type
0C8A7      SF_pt
0C8C8      CF_pt
0C8DC      boolean_FS?_pt
0E3E2   !  IF
0E3F8   !  THEN_if
0E459   !  ELSE_iferr
0E459   !  ELSE_if
0E479   !  END_if
0E479   !  END_iferr
0E4D2   !  WHILE
0E4F2   !  REPEAT_while
0E535   !  DO
0E555   !  UNTIL_do
0E56B   !  START
0E5B3   !  FOR
0E60F   !  NEXT_for
0E60F   !  NEXT_start
0E73E   !  STEP_for
0E73E   !  STEP_start
0E77A   !  IFERR
0E842      HALT
0E862   !  program
0E87D   !  →
0E9B5   !  »_local
0E9D0   !  «
0E9D0   !  «_local
0E9E6   !  »
0E9FC   !  begin_quoted_name
0EA17   !  end_quoted_name
0EA2D   !  END_while
0EA4D   !  END_do
0EA90   !  THEN_iferr
0EAC5      list_{ 'noname 'stop }
0EB47      list_{ st ofs tok }
0EBC3      list_{ }
0EC68      list_{ }
0FA82      list_{ { } }
0FFCB      string_"HALT"
0FFDD      string_"ELSE"
0FFEF      string_"END"
0FFFF      string_"UNTIL"
10013      string_"REPEAT"
10029      string_"NEXT"
1003B      string_"STEP"
1004D      string_"IF"
1005B      string_"DO"
10069      string_"IFERR"
1007D      string_"START"
10091      string_"FOR"
100A1      string_"THEN"
100B3      string_"→"
100BF      string_"WHILE"
102D5      string_"bodh"
102F6      list_{ pt_2 pt_8 pt_10 pt_16 }
1054E      list_{ 'ofs 'tok }
```

```
10C2F       binary_#0
11001   !   array_errors3xx
112B4       real_0
112C9       real_1
112DE       real_2
112F3       real_3
11308       real_4
1131D       real_5
11332       real_6
11347       real_7
1135C       real_8
11371       real_9
11386       real_-1
1139B       real_-2
113B0       real_-3
113C5       real_-4
113DA       real_-5
113EF       real_-6
11404       real_-7
11419       real_-8
1142E       real_-9
11443       real_3.14159265359
11472       real_9.99999999999E499
11487       real_-9.99999999999E499
1149C       real_1.E-499
114B1       real_-1.E-499
11596       pt_57
115A0       pt_58
115AA       pt_59
115B4       pt_60
115BE       pt_61
115C8       pt_62
115D2       pt_63
115DC       pt_64
11900       D→R_real
11933       R→D_real
11951       →HMS_real
11956       real_3600
1197F       HMS→_real
119A2       HMS+_1real2real
119CA       HMS-_1real2real
11A01       MAX_1real2real
11A01       MAX_1complex2complex
11A24       MIN_1real2real
11A24       MIN_1complex2complex
11AEF       boolean_==0?_real
11C59       SIGN_real
11C83       ABS_real
11CA4       NEG_real
11CB6       MANT_real
11CEB       +_1real2real
11D1D       -_1real2real
11D48       *_1real2real
11D5F       %_1real2real
11D82       /_1real2real
11D99       %T_1real2real
11DB3       %CH_1real2real
11E2D       INV_real
11E83       EXP_real
11E99       EXPM_real
11EFF       LNP1_real
11F13       ALOG_real
11F38       MOD_1real2real
11F5D       SIN_real
11FAC       COS_real
11FEC       TAN_real
12060       ATAN_real
120F7       SINH_real
12120       COSH_real
12134       TANH_real
12158       ASINH_real
12194       XPON_real
121C0       COMB_1real2real
121D6       PERM_1real2real
12450       FP_real
12474       IP_real
12485       CEIL_real
124A6       FLOOR_real
124E6       RAND_do
1256A       RDZ_real
125A1       RDZ_real_notclock
125F6       FACT_real
12600       real_260
1266B       real_-260
14296       ROOT_1complex2name3algebraic
14296       ROOT_1complex2name3program
14296       ROOT_1list2name3algebraic
14296       ROOT_1list2name3program
14296       ROOT_1real2name3algebraic
14296       ROOT_1real2name3program
142BE       list_{ 'nohalt }
14345       list_{ pt_14 }
14363       list_{ " " " " " " " " }
14FAC       pt_2565
15006       pt_2566
15024       pt_287
15042       pt_2561
15056       pt_2562
1510A       pt_2564
154CA       string_"+  "
154DA       string_"-  "
154F4       string_"?  "
156E9       pt_2563
161CB       list_{ '#c' }
1680F       list_{ " }
16B9F       binary_#0
17970       binary_#0
1797F       binary_#1
1798E       binary_#2
1799D       binary_#3
179AC       binary_#4
17D53   !   array_errorsAxx
17E1C   !   array_errors1xx
182C0       pt_262
182CA       pt_263
182D4       pt_640
182DE       pt_32896
182E8       pt_131200
182F2       pt_1152
182FC       pt_2176
18306       pt_4224
18310       pt_128
1831A       pt_68
18324       pt_32832
18371       Error_102
18381       Error_103
18391       Error_104
183A1       Error_106
183B1       Error_107
183C1       Error_11E
183D1       Error_11F
183E1       Error_120
183F1       Error_121
18401   !   Error_122
18411   !   Error_123
18421       Error_124
18431       Error_125
18441       Error_126
18451       Error_128
18461       Error_129
18471       Error_12A
18481       Error_12B
184A7       CMD_on
184B8       CMD_offbutnoclear
184C9       CMD_off
184DD       PURGE_commandstack
18506       boolean_TRAC=off?
1851F       boolean_LAST=on?
18533       LAST_on
18558       UNDO_on
18569       UNDO_off
186AC       MENUS_on
186BD       MENUS_off
186CE       boolean_MENUS=on?
18756       Entrymode_algebraic_off
18767       Entrymode_algebraic_on
18778       reverse_video_on
18789       reverse_video_off
1879A       boolean_LC=on?
187AB       LC_on
187BC       LC_off
188A2       INS_on
188B3       INS_off
188C4       INS_on
1891A       await_key_when_done
189E7       reverse_level1_when_done
18A85       set_mf
18C79       list_{ boop boop boop boop boop boop }
18D50       string_" Low Memory!"
18E58       off
18E7B       pt_3839
18F02       pt_260
1903D       string_"<>"]])≤≥→Σ˚u"
19069       list_{ pt_1 pt_2 pt_7 pt_8 pt_9 }
191BD       string_"«=laYZ#[[(STUVWXM
            NOPQRGHIJKLABCDEF"
1922D       pt_64
19241       pt_91
195D4       EDITe_string
19769   !   off&hop
19A10       SHIFT_key
19A1A       string_"SHIFT"
19A47       SHIFT_keyshifted
19AAE       ALFA_key
19B8A       VISIT_key
19BA3       VISIT_keydo
19BAD       string_"VISIT"
19C75       DUP_key
19CB1       string_"ENTER"
1A064       COMMAND_key
1A082       COMMAND_keydo
1A08C       string_"COMMAND"
1A0E0       NEG_key
1A0EA       string_"CHS"
1A12C       EEX_key
1A136       string_"EEX"
1A16E       WAIT_real
1A173       real_8192
1A1F6       KEY_do
1A214       pt_scancode→KEY
1A237       pt_scancode→KEYstring
1A30E       string_firstrowkey_pt
1A313       list_{  "INS"  "DEL"  "UP"
            "DOWN" "LEFT" "RIGHT" }
1A318       string_"INS"
1A328       string_"DEL"
1A338       string_"UP"
1A346       string_"DOWN"
1A358       string_"LEFT"
1A36A       string_"RIGHT"
1A432       CR_do
1A473       PRST_do
1A4E1       string_"[Empty Stack]"
1A505       PR1_do
1A67C       PRSTC_do
1A6C2       string_number+": "_pt
1A899       list_{ 146 }
1A8FB       list_{ 1 }
1A928       →STR_formatted_complex
1A987       string_"  "
1A9AD       string_"Row "
1A9BF       PRUSR_do
1A9D8       VARS_do
1AA28       PRVAR_name
1AA6E       PRMD_do
1AA73       string_"Format "
1AA95       string_" Base "
1AAC9       string_"RADIANS"
1AAE1       string_"DEGREES"
1AB08       string_" Radix "
1AB47       string_"Undo"
1AB6D       string_"Command"
1AB99       string_"Last"
1ABB5       string_"Multiline"
1ABFD       string_format
1AC43       string_format_pt
1AC84       string_base
1ACCA       +"ON/OFF"_1boolean2string
1ACCF       string_" ON "
1ACE1       string_" OFF"
1AD02       +"  "_string
1AD16       PRLCD_do
1ADF2       absoluteUNPIXEL_1pt2pt
1ADFE       absolutePIXEL_1pt2pt
1B362       string_line1
1B372       string_line2
1B382       string_line3
1B392       string_line4
1BA31       label1_string
1BB44       label2_string
1BB57       label3_string
1BB6A       label4_string
1BB7D       label5_string
1BB90       label6_string
1BBA3       DISP_1_string
1BBED       DISP_2_string
1BC00       DISP_3_string
1BC13       DISP_4_string
1BC79       DISP_1any2real3string
1BC91       DISP_1pt2string
1BCC1       CURSOR_key
1BCCB       string_"CURSOR"
1BD27       menu_off
1BF84       EDIT_key
1BF9D       EDIT_keydo
1BFA7       string_"EDIT"
1BFEB       ALFA_on
1C009       EDIT_any
1C045       EDITn_string
1C23B       EDITne_string
1C5DF       pt_GET_1_list
1CCF6       pt_22
1CD37       BACK_key
1CD41       string_"BACK"
1CD58       pt_4128
1D94F       USER_key1
```

```
1D96D   USER_key2
1D98B   USER_key3
1D9A9   USER_key4
1D9C7   USER_key5
1D9E5   USER_key6
1DA03   USER_key_pt
1DA21   boolean_exists?&RCLifso_pt
1DAEE   USER_label1
1DB02   USER_label2
1DB16   USER_label3
1DB2A   USER_label4
1DB3E   USER_label5
1DB52   USER_label6
1DB66   USER_label_pt
1DC92   CLUSR_do
1DE0D   RCL_'EQ'
1DE26   SOLVR_key
1DE30   string_"SOLVR"
1DFB6   name_'EXPR='
1DFC7   name_'LEFT='
1DFD8   name_'RT='
1DFE5   string_" "
1DFF1   SOLVR_key1
1E00F   SOLVR_key2
1E02D   SOLVR_key3
1E04B   SOLVR_key4
1E069   SOLVR_key5
1E087   SOLVR_key6
1E0A5   SOLVR_key_pt
1E14F   list_{ a b }
1E1A2   string_": "
1E246   string_"EXPR="
1E28C   string_"LEFT="
1E2C8   list_{ right }
1E30D   string_"RIGHT="
1E337   list_{ solvid badlist }
1E3FA   list_{ 'nohalt }
1E413   string_"SOLVING FOR "
1E50E   list_{ msg }
1E6C4   STEQ_any
1E6E2   boolean_EQexists?&RCLifso
1E737   boolean_exists?&RCLifso_name
1E791   name_''
1E798   name_'EQ'
1E7A3   →NUM_array
1ED41   FIX_pt
1ED4D   SCI_pt
1ED59   ENG_pt
1ED65   STD_do
1EE18   boolean_RDX,=on?
1EE55   string_+": "_pt
1EE61   pt→string
1F151   PURGEassist
1F24B   pt_84
1F290   string_"Custom Menu"
1F2BF   PURGEassist_stack
1F2D8   PURGEassist_commandstack
1F2F1   PURGEassist_undostack
1F323   PURGEassist_lastarguments
1F718   PURGE_stack
1F859   annunciator_BUSY_on
1F866   annunciator_BUSY_off
1F873   ALFA_lock
1F880   ALFA_off
1F88D   annunciator_BAT_on
1F89A   annunciator_BAT_off
1F8A7   SHIFT_on
1F8B4   SHIFT_off
1F8C1   annunciator_(2PI)_on
1F8CE   annunciator_(2PI)_off
1F8DB   annunciator_HALT_on
1F8E8   annunciator_HALT_off
1F8F5   annunciator_PRINT_on
1F902   annunciator_PRINT_off
1F9DE   PURGE_custommenu
1FBA7   edit_up
1FC88   edit_right
1FD4D   list_custommenu
1FE22   MENU_display_pt
1FEEA   clear_menutoggle
1FF0A   menu_on
2033F   PREV_key
20099   MENU_pt
200B7   DUP&MENU_display_pt
20120   MENUactivate_pt
20256   NEXT_key
20260   string_"NEXT"
20583   MENU_list
206DC   MENU_real
207AE   pt_254
20C24   pt_524288
20CB0   string_"E"
20CBC   string_"IE"
20DE0   PURGE_lastarguments
20E8C   string_" Error:"
21094   PR_string
2123E   list_{ lists_of_all_menus }
212F2   list_{ }
212FC   list_{ }
21306   USER_menu
21310   string_"USER"
21331   list_user_menu
21359   SOLVR_menu
2136D   list_solver_menu
21395   BRANCH_menu
213A9   list_{ IF IFERR THEN ELSE
        END boop START FOR
        NEXT STEP IFT IFTE ....
2140D   LOGS_menu
2141A   list_{ LOG ALOG LN EXP LNP1
        EXPM SINH ASINH COSH
        ACOSH TANH ATANH }
2146D   list_{ Σ+ Σ- NΣ CLΣ STOΣ
        RCLΣ TOT MEAN SDEV
        VAR MAXΣ MINΣ COLΣ ....
214EF   MODE_menu
214FC   list_{ STD_key FIX_key
        SCI_key ENG_key DEG_key
        RAD_key CMD_toggle ....
21542   TRIG_menu
2154C   string_"TRIG"
2156D   list_{ SIN ASIN COS ACOS
        TAN ATAN P→R R→P R→C
        C→R ARG boop ....
215DE   list_{ DUP OVER DUP2 DROP2
        ROT LIST→ ROLLD PICK
        DUPN DROPN DEPTH ....
21631   list_{ COLCT EXPAN SIZE
        FORM OBSUB EXSUB TAYLR
        ISOL QUAD SHOW ....
21677   SOLV_menu
21681   string_"SOLV"
216A2   list_{ STEQ RCEQ SOLVR_key
        ISOL QUAD SHOW ROOT }
216F2   BINARY_menu
21706   list_{ DEC_key HEX_key
        OCT_key BIN_key STWS
        RCWS RL RR RLB RRB ....
2178B   list_{ →LIST LIST→ PUT GET
        PUTI GETI POS SUB SIZE }
217CF   list_{ NEG FACT RAND RDZ
        MAXR MINR ABS SIGN MANT
        XPON boop boop IP ....
21854   list_{ SST_key HALT_key
        ABORT KILL WAIT KEY
        BEEP CLLCD DISP CLMF ....
2189A   TEST_menu
218AE   list_{ SF CF FS? FC? FS?C
        FC?C AND OR XOR NOT
        SAME == STOF RCLF TYPE
        }
21910   list_{ STO+ STO- STO* STO/
        SNEG SINV SCONJ }
2194A   list_{ PR1 PRST PRVAR
        PRLCD CR TRAC_toggle
        PRSTC PRUSR PRMD }
2198E   list_{ →ARRY ARRY→ PUT
        GET PUTI GETI SIZE RDM
        TRN CON IDN RSD ....
21A1D   list_{ R→C C→R RE IM
        CONJ SIGN R→P P→R ABS
        NEG ARG }
21A6B   list_{ →STR STR→ CHR
        NUM →LCD LCD→ POS
        SUB SIZE DISP }
21AA7   PLOT_menu
21ABB   list_{ STEQ RCEQ PMIN
        PMAX INDEP DRAW_key
        PPAR_key RES AXES
        CENTR ....
21B3D ! list_{ COLCT_form
        EXPAN_form LEVEL_form
        EXGET_form [<-]_form ....
21B65 ! list_{ DNEG_form DINV_form
        *1_form /1_form ^1_form
        +1-1_form }
21B8D ! list_{ -()_form <-→_form
        <-M_form M→_form <-A_form
        A→_form }
21BB5 ! list_{ AF_form }
21BC4 ! list_{ 1/()_form <-→_form
        <-D_form D→_form <-A_form
        A→_form }
        21BEC!list_{ -()_form L()_form
        <-M_form M→_form }
21C0A ! list_{ 1/()_form E()_form
        <-D_form D→_form <-A_form
        A→_form }
21C32 ! list_{ 1/()_form E^_form boop
        D→_form }
21C50 ! list_{ -()_form L*_form boop
        D→_form }
21C6E ! list_{ →()_form }
21C8A   list_{ MEM MENU ORDER
        PATH HOME CRDIR VARS
        CLUSR_key }
21CBC   PPAR_key
21CC6   string_"PPAR"
21D05   DRWΣ_key
21D0F   string_"DRWΣ"
21D2B   STD_key
21D49   FIX_key
21D62   SCI_key
21D7B   ENG_key
21D94   DEG_key
21DB2   RAD_key
21DD0   RDX,_toggle
21E02   CMD_toggle
21E2F   LAST_toggle
21E5C   UNDO_toggle
21E7A   UNDO_off
21EA7   ML_toggle
21ED4   ML_on
21EE8   ML_off
21F01   label_STD
21F06   string_"STD"
21F2A   label_FIX
21F2F   string_"FIX"
21F53   label_SCI
21F58   string_"SCI"
21F7C   label_ENG
21F81   string_"ENG"
21FA5   label_DEG
21FAA   string_"DEG"
21FC9   label_RAD
21FCE   string_"RAD"
21FED   label_RDX,
21FF2   string_"RDX,"
22040   label_CMD
22045   string_"CMD"
2206E   label_LAST
22073   string_"LAST"
2209E   label_UNDO
220A3   string_"UNDO"
220D3   label_ML
220D8   string_"ML"
220FF   DEC_key
2211D   HEX_key
2213B   OCT_key
22159   BIN_key
22177   label_DEC
2217C   string_"DEC"
221A0   label_HEX
221A5   string_"HEX"
221C9   label_OCT
221CE   string_"OCT"
221F2   label_BIN
221F7   string_"BIN"
2221B   SST_key
22234   label_SST
22239   string_"SST"
2226C   HALT_key
22280   string_"HALT"
2229C   label_HALT
222BA   TRAC_toggle
222F6   string_"TRAC"
22344   string_" "
22350   string_""
223DB   pt_format
2240F   pt_digits
2242C   boop
22459   CUSTOM_key
2246D   CLUSR_key
22477   string_"CLUSR"
2249F   STO_key
224A9   string_"STO"
224C8   RCL_key
224DC   X² key
224F0   √X key
22504   ^_key
22518   INV_key
2252C   +_key
22540   -_key
22568   *_key
2257C   SWAP_key
```

```
22590  DROP_key
225A4  EVAL_key
225AE  string_"EVAL"
225CF  →NUM_key
225E3  ROLL_key
225F7  PURGE_key
2260B  CLEAR_key
2261F  CONT_key
22633  string_"CONTINUE"
22661  UNDO_key
22666  string_"UNDO"
22691  CONVERT_key
226A5  d/dx_key
226B9  %CH_key
226CD  %_key
226E1  LAST_key
226F5  ∫ key
2276B  pt_errn
22790  ERRN_do
227A9  ERRM_do
227BD  →STR_any
227DB  DISP_1real2any
2283A  POS_1string2string
228F5  RND_real
229C3  CHR_real
229E6  NUM_string
22A22  STR→_string
22A45  BEEP_1real2real
22AA7  BEEP_1pt_Hz2pt_s
22BE0  boing
22C0F  >_1string2string
22CAE  <_1string2string
22CC2  ≥_1string2string
22CEA  ≤_1string2string
22D12  local_'nohalt
22D3B  list_{ 'halt }
22D40  local_'halt
22DE7  pt_291
22E05  pt_290
22E55  lasthaltedprogram_intoUNDObuffer
22EBE  KILL_do
22F1D  SST_do
23016  HALT&SST_any
2385A  DISP_1_reverse_string
23AF9  EDIT_chr_pt
23CB0  '_type
23CBA  string_""
23CE6  CONVERT_type
23CEB  string_"CONVERT"
23D0D  CLEAR_type
23D12  string_"CLEAR"
23D30  SWAP_type
23D35  string_"SWAP"
23D51  STO_type
23D56  string_"STO"
23D70  PURGE_type
23D75  string_"PURGE"
23D93  ROLL_type
23D98  string_"ROLL"
23DB4  DROP_type
23DB9  string_"DROP"
23DD5  /_type
23DEB  *_type
23E01  -_type
23E17  +_type
23E2D  %_type
23E5C  %CH_type
23E61  string_"%CH"
23E80  INV_type
23E85  string_"INV"
23EA4  X²_type
23EA9  string_"X²"
23EC6  ^_type
23EDC  √X type
23EF2  LAST_type
23EF7  string_"LAST"
23F13  RCL_type
23F18  string_"RCL"
23F32  EVAL_type
23F37  string_"EVAL"
23F53  ∫ type
23F69  »_type
23F7F  π_type
23F95  ?_type
23FAB  ,_type
23FC1  ._type
23FD7  0_type
23FED  1_type
24003  2_type
24019  3_type
2402F  4_type
24045  5_type
2405B  6_type
24071  7_type
24087  8_type
2409D  9_type
240B3  d/dx_type
240C9  →NUM_type
240CE  string_"→NUM"
240EA  ≠_type
24277  name_'constant'
242AC  name_'PPAR'
2430B  complex_(-6.8, -1.5)
24330  complex_(6.8, 1.6)
2435A  complex_(0, 0)
24474  string_" Not In Equation"
244BC  string_"Constant Equation"
244E8  string_"Using "+_name
244ED  string_"Using "
24530  PMIN_any
24544  PMAX_any
24558  INDEP_any
2456C  RES_any
24580  AXES_any
245CB  *W_real
2466B  *H_real
24693  DRAW_do
246CF  PIXEL_complex
247C4  CLLCD_do
247DD  DRAX_do
2484B  makePPAR&LIST→&DROP
2487D  CENTR_complex
24927  SWAP&C→R2_1complex2complex
249C9  DRAW_key
249D3  string_"DRAW"
24AE4  makePPAR&constant_returnR
       ES&AXES
24C4C  pt_256
25133  LCD→_do
25174  RIGHT_3_string
2518D  →LCD_string
251A1  →LCD_do_string
25223  DGTIZ_do
25DAB  string_"2: "
25DCA  string_"3: "
25DDA  string_"4: "
25F4D  string_"1: "
25F8A  string_" "
25FA4  VIEWup_key
26233  VIEWdown_key
26242  pt_1056
26436  boolean_ML=on?
2644A  alternate_stack_display
26477  DISP_1_"No Room to Show Stack"
2647C  string_"No Room to Show Stack"
26569  string_"Directory"
26CE8  string_"}"
26DF8  string_"{"
27634  string_" [ "
2774D  string_"[[ "
27780  string_"]]"
27E69  list_{ rightkeys, ALFA }
27F13  list_{ SHIFTrightkeys, DIRECT }
27FBD  list_{ SHIFTrightkeys, ALFA }
28067  PATH_do
280BC  HOME_do
280DA  CRDIR_do
28175  boolean_exists?DUP&RCLifso_name
28215  AND_1string2string
28229  OR_1string2string
2823D  XOR_1string2string
28288  NOT_string
283B7  UTPN_1real2real3real
2867B  UTPC_1real2real
28953  UTPF_1real2real3real
2905D  UTPT_1real2real
296A1  STO&_any
296AB  name_'&DAT'
296C4  CL&_do
296DD  RCL&_do
29737  &+_real
2974B  list_{ pt_1 pt_1 }
297CD  Σ+_array
298FE  Σ-_do
29A52  NΣ_do
29A75  MAXΣ_do
29A8E  MEAN_do
29AA7  MINΣ_do
29AC0  SDEV_do
29AD9  TOT_do
29AF2  VAR_do
29BBA  COLΣ_1real2real
29BEC  create&RCL&LIST→_&PAR
29BF6  name_'&PAR'
29C0F  create&RCL_&PAR
29C14  list_{ 1 2 0 0 }
29C96  pt_4369
29CF5  CORR_do
29D86  COV_do
29DE0  SCLΣ_do
29EB2  DRWΣ_do
2A083  LR_do
2A141  PREDV_real
2A739  !  array_errors6xx
2A7FA  !  array_errors5xx
2AF1C  CON_1real2array
2AFAD  IDN_array
2B064  NEG_array
2B0B9  RND_array
2B10E  RND_complex
2B136  CONJ_array
2B1A9  RE_array
2B217  IM_array
2B30C  C→R_array
2B389  +_1array2array
2B4F1  -_1array2array
2B76C  *_1array2complex
2B76C  *_1complex2array
2B76C  *_1real2array
2B884  /_1complex2array
2B884  /_1real2array
2B9BF  X²_array
2B9C9  *_1array2array
2BAAF  list_{ " " " " }
2BB59  RSD_1array2array3array
2BD2F  DOT_1array2array
2BDD9  CROSS_1array2array
2BDED  list_{ pt_3 }
2BFC3  list_{ 3 }
2BFC8  real_3
2BFEC  RNRM_array
2C0A5  CNRM_array
2C154  ABS_array
2C1C2  DET_array
2C2E4  INV_array
2C348  /_1array2array
2C3E8  list_{ '#a' '#b' " }
2C4E6  list_{ '#b' " }
2C551  list_{ " " }
2C579  list_{ " " " }
2D89C  pt_1281
2DD23  TRN_array
2DF3E  CONVERT_1real2any3any
2E737  UNITS_key
2E75A  pt_23
2E7D7  string_""
2EA4D  unit_description_1pt2any
2EDBA  string_"kg"
2EDC8  string_"m"
2EDD4  string_"A"
2EDE0  string_"s"
2EDEC  string_"°K"
2EDFA  string_"cd"
2EE08  string_"mol"
2EE18  string_"?"
2FEF0  binary_#0
2FF0A  binary_#100000000
2FFA6  pt_123
30121  !  array_errorsBxx
301E7  complex_(-1, 0)
3020C  IM_real
30220  RE_complex
30234  IM_complex
302AA  P→R_real
30417  NEG_complex
30459  CONJ_complex
30468  +_1real2complex
3047C  +_1complex2real
3049F  +_1complex2complex
30544  -_1complex2real
30558  -_1real2complex
3056C  -_1complex2complex
305BC  *_1real2complex
305D0  *_1complex2real
305FD  *_1complex2complex
30684  /_1complex2real
306F2  /_1real2complex
30738  /_1complex2complex
3079C  INV_complex
3095E  ABS_complex
30995  ARG_complex
309AE  P→R_complex
309EF  R→P_complex
30A1C  SIGN_complex
30A67  √X complex
30B02  EXP_complex
```

```
30B57  LN_complex
30C38  LOG_complex
30C88  ALOG_complex
30CCF  ^_1complex2real
30CED  ^_1real2complex
30D01  ^_1complex2complex
30E5A  complex_(0, 0)
30EA7  complex_(1, 0)
30EE0  SIN_complex
30F26  COS_complex
30F71  TAN_complex
30FFD  SINH_complex
31016  COSH_complex
3102A  TANH_complex
31043  ATAN_complex
3104D  complex_(0, 1)
3107C  complex_(0, -1)
311CD  ATANH_complex
311E6  ASIN_complex
31204  ASINH_complex
3121D  ACOSH_complex
3124A  ACOS_complex
31680  HEX_do
3168C  BIN_do
31698  OCT_do
316A4  DEC_do
316E3  STWS_real
316F7  STWS_pt_real
3174C  RCWS_do
31760  AND_1binary2binary
31771  OR_1binary2binary
31782  XOR_1binary2binary
317AA  NOT_binary
317BA  SL_binary
317CA  SLB_binary
317DD  SR_binary
317ED  SRB_binary
31800  RR_binary
3183D  RRB_binary
31868  RL_binary
31897  RLB_binary
318C1  ASR_binary
31903  +_1binary2binary
31913  -_1binary2binary
31926  *_1binary2binary
31958  /_1binary2binary
31A84  pt_ws
31AA8  pt_base
31ABC  string_ascii→binary_string
31AD5  byte_basepostfix
31B28  string_ascii→based_string
31D64  /_1binary2real
31D82  /_1real2binary
31D96  *_1binary2real
31DAF  *_1real2binary
31DC3  -_1binary2real
31DE1  -_1real2binary
31DF5  +_1binary2real
31E0E  +_1real2binary
31E22  B→R_binary
31EC3  R→B_real
31FAA  ==_1binary2binary
31FBD  ≠_1binary2binary
31FD1  >_1binary2binary
32000  ≥_1binary2binary
32013  ≤_1binary2binary
32026  <_1binary2binary
32B28  IFTE_1algebraic2algebraic3algebraic
32B28  IFTE_1algebraic2complex3algebraic
32B28  IFTE_1algebraic2real3algebraic
32B28  IFTE_1complex2algebraic3algebraic
32B28  IFTE_1complex2complex3algebraic
32B28  IFTE_1complex2real3algebraic
32B28  IFTE_1real2algebraic3algebraic
32B28  IFTE_1real2complexalgebraic
32B28  IFTE_1real2real3algebraic
32B2D  list_{ 'tcls 'fcls }
32E96  STO_algebraic
32F45  list_{ 'num }
32F90  d/dx_1name2algebraic
33017  list_{ 'dvar }
33362  MINR_do
3339E  MAXR_do
333C1  PI_do
333E4  i_do
33407  e_do
3347A  list_{ 'xSYMfcn 'xfcn }
33572  RE_algebraic
3358B  IM_algebraic
335A4  NOT_algebraic
335BD  NEG_algebraic
335D6  ABS_algebraic
335EF  CONJ_algebraic
33608  INV_algebraic
33621  ARG_algebraic
3363A  P→R_algebraic
33653  R→P_algebraic
3366C  SIGN_algebraic
33685  √X algebraic
3369E  X²_algebraic
336B7  SIN_algebraic
336D0  COS_algebraic
336E9  TAN_algebraic
33702  SINH_algebraic
3371B  COSH_algebraic
33734  TANH_algebraic
3374D  ASIN_algebraic
33766  ACOS_algebraic
3377F  ATAN_algebraic
33798  ASINH_algebraic
337B1  ACOSH_algebraic
337CA  ATANH_algebraic
337E3  EXP_algebraic
337FC  LN_algebraic
33815  LOG_algebraic
3382E  ALOG_algebraic
33847  LNP1_algebraic
33860  EXPM_algebraic
33879  FACT_algebraic
33892  IP_algebraic
338AB  FP_algebraic
338C4  FLOOR_algebraic
338DD  CEIL_algebraic
338F6  XPON_algebraic
3390F  MANT_algebraic
33928  D→R_algebraic
33941  RND_algebraic
3395A  R→D_algebraic
33DBA  list_{ 'oth }
33EEB  list_{ 'scl 'xSYMfcn 'xfcn }
33F72  list_{ 'xSYMfcn 'xfcn }
340EE  =_1any2any
34161  AND_1real2algebraic
3417A  AND_1algebraic2real
34193  AND_1algebraic2algebraic
341AC  OR_1real2algebraic
341C5  OR_1algebraic2real
341DE  OR_1algebraic2algebraic
341F7  XOR_1real2algebraic
34210  XOR_1algebraic2real
34229  XOR_1algebraic2algebraic
34242  ==_1complex2algebraic
34242  ==_1real2algebraic
3425B  ==_1algebraic2complex
3425B  ==_1algebraic2real
34274  ==_1algebraic2algebraic
3428D  ≠_1complex2algebraic
3428D  ≠_1real2algebraic
342A6  ≠_1algebraic2complex
342A6  ≠_1algebraic2real
342BF  ≠_1algebraic2algebraic
342D8  <_1real2algebraic
342F1  <_1algebraic2real
3430A  <_1algebraic2algebraic
34323  >_1algebraic2algebraic
34323  >_1real2algebraic
3433C  >_1algebraic2real
3436E  ≤_1real2algebraic
34387  ≤_1algebraic2real
343A0  ≤_1algebraic2algebraic
343B9  ≥_1real2algebraic
343D2  ≥_1algebraic2real
343EB  ≥_1algebraic2algebraic
34404  %_1real2algebraic
3441D  %_1algebraic2real
34436  %_1algebraic2algebraic
3444F  %CH_1real2algebraic
34468  %CH_1algebraic2real
34481  %CH_1algebraic2algebraic
3449A  %T_1real2algebraic
344B3  %T_1algebraic2real
344CC  %T_1algebraic2algebraic
344E5  MAX_1real2algebraic
344FE  MAX_1algebraic2real
34517  MAX_1algebraic2algebraic
34530  MIN_1real2algebraic
34549  MIN_1algebraic2real
34562  MIN_1algebraic2algebraic
3457B  ^_1complex2algebraic
3457B  ^_1real2algebraic
34594  ^_1algebraic2complex
34594  ^_1algebraic2real
345AD  ^_1algebraic2algebraic
345C6  +_1real2algebraic
345DF  +_1algebraic2complex
345DF  +_1algebraic2real
345F8  +_1algebraic2algebraic
34611  -_1complex2algebraic
34611  -_1real2algebraic
3462A  -_1algebraic2complex
3462A  -_1algebraic2real
34643  -_1algebraic2algebraic
3465C  *_1real2algebraic
34675  *_1algebraic2complex
34675  *_1algebraic2real
3468E  *_1algebraic2algebraic
346A7  /_1complex2algebraic
346A7  /_1real2algebraic
346C0  /_1algebraic2complex
346C0  /_1algebraic2real
346D9  /_1algebraic2algebraic
346F2  MOD_1real2algebraic
3470B  MOD_1algebraic2real
34724  MOD_1algebraic2algebraic
34F99  list_{ 2 ^ / }
35084  list_{ CONJ * + }
350AC  list_{ ABS 2 * / }
3510B  list_{ X² - √X INV NEG }
35156  list_{ X² 1 - √X / }
35205  list_{ CONJ - 2 I * / }
3525F  list_{ X² - √X INV }
352B4  list_{ X² + √X / }
352F5  list_{ X² + INV }
35345  list_{ X² - / }
354AD  list_{ 1 + / }
354E9  list_{ 10 LN * / }
35651  list_{ √X * / }
35692  list_{ TAN X² + }
356CE  list_{ COSH X² / }
357AA  list_{ 1 - ^ * }
358B8  list_{ PI 180 / * }
358E5  list_{ 180 PI / * }
35921  list_{ 'dv }
35960  algebraic_'0'
3596F  algebraic_'1/0'
35D5C  list_{ 'dv 'op 'nm }
35DF3  ISOL_1name2algebraic
364AE  list_{ 10 LN / + }
3661B  list_{ PI I * }
36652  list_{ 2 PI * I * }
367E7  list_{ 'nl 'ns }
36858  name_'n0'
36876  name_'s0'
36925  EXPAN_algebraic
36925  EXPAN_complex
36925  EXPAN_real
36CBD  COLCT_algebraic
36CBD  COLCT_complex
36CBD  COLCT_real
36DEE  list_{ '*s }
37080  list_{ '+s }
37D83  SHOW_1name2algebraic
37DBF  list_{ 'fl }
37FA2  QUAD_1name2algebraic
3804C  list_{ 'c 'b 'a }
38090  algebraic_'s1'
38095  name_'s1'
38140  list_{ 'n 'prog }
38300  TAYLR_1real2name3algebraic
3837D  list_{ 'n }
383F6  ∫_1real2name3algebraic
3847D  list_{ 'n }
3852F  FORM_algebraic
3852F  FORM_complex
3852F  FORM_real
385F2  list_{ 'maniplams 'maniplams
       'maniplams 'maniplams
       'maniplams ....
38B45  !  LEVEL_form
38C0C  !  [<-]_form
38C75     string_"[<-]"
38C85  !  [→]_form
38CC6     string_"[→]"
38CD6  !  EXGET_form
38CE0     string_"EXGET"
38E89  !  DNEG_form
38E93     string_"DNEG"
38EC8  !  DINV_form
38ED2     string_"DINV"
38F07  !  *1_form
38F11     string_"*1"
38F42  !  ^1_form
38F4C     string_"^1"
38F7D  !  /1_form
38F87     string_"/1"
38FB8  !  +1-1_form
```

```
38FC2       string_"+1-1"
38FD9       list_{ 1 + 1 - }
39073   !   EXPAN_form
3907E       string_"EXPAN"
390B5   !   COLCT_form
390BF       string_"COLCT"
390F6   !   <-→_form
39100       string_"<-→"
3911D   !   <-A_form
39127       string_"<-A"
39144   !   A→_form
3914E       string_"A→"
3916B   !   AF_form
39175       string_"AF"
39192   !   M→_form
3919C       string_"M→"
391B9   !   <-M_form
391C3       string_"<-M"
391E0   !   -()_form
391EA       string_"-()"
39209   !   1/()_form
39213       string_"1/()"
39234   !   E()_form
3923E       string_"E()"
3925D   !   L()_form
39267       string_"L()"
39286   !   L*_form
39290       string_"L*"
392AD   !   E^_form
392B7       string_"E^"
392D4   !   →()_form
392DE       string_"→()"
392FD   !   <-D_form
39307       string_"<-D"
39324   !   D→_form
3932E       string_"D→"
3934B       EXGET_1real2algebraic
39378       OBGET_1real2algebraic
39463       EXSUB_1algebraic2real3algebraic
39463       EXSUB_1complex2real3algebraic
39463       EXSUB_1real2real3algebraic
394B3       OBSUB_1list2real3algebraic
3975B       SIZE_algebraic
3CEDC       string_""
3D5B9       DROP3..3_1any2any3any
3D5D5       REVERSE3_1any2any3any
3D5F6       DROP3_1any2any3any
3D5FF       DROP4_1any2any3any4any
3D61C       DROP2..2_1any2any
3D636       ROLLD3_1any2any3any
3D65C       ROLL4_1any2any3any4any
3D682       ROLL5_1any2any3any4any5any
3D6B8       ROLL6_1any2any3any4any5any6any
3D6F2       ROLL7_1any2any3any4any5any6any7any
3D72C       ROLLD4_1any2any3any4any
3D752       ROLLD5_1any2any3any4any5any
3D788       ROLLD6_1any2any3any4any5any6any
3D7C2       DROP2..3_1any2any3any
3D7DE       DROP2..4_1any2any3any
3D7F8       PICK3_1any2any3any
3D815       PICK4_1any2any3any4any
3DC5F   !   IF_boolean
3E9D7       GET_1pt2list
3E9EB       +" "_string
3EB96       pt_45
3EBA0       pt_48
3EBAA       pt_50
3EBB4       pt_51
3EBBE       pt_52
3EBC8       pt_53
3EBD2       pt_55
3EBDC       pt_65
3EBE6       pt_66
3EBF0       pt_67
3EBFA       pt_68
3EC04       pt_69
3EC0E       pt_70
3EC18       pt_80
3EC22       pt_81
3EC2C       pt_82
3EC36       pt_83
3EC40       pt_84
3EC4A       pt_85
3EC54       pt_86
3EC5E       pt_96
3EC68       pt_97
3EC72       pt_98
3EC7C       pt_100
3EC86       pt_101
3EC90       pt_112
3EC9A       pt_113
3ECA4       pt_117
3ECAE       pt_261
3ECB8       pt_136
3ECC2       pt_137
3ECCC       pt_161
3ECD6       pt_162
3ECE0       pt_166
3ECEA       pt_167
3ECF4       pt_169
3ECFE       pt_170
3ED08       pt_177
3ED12       pt_187
3ED1C       pt_221
3ED26       pt_222
3ED30       pt_237
3ED3A       pt_238
3ED44       pt_256
3ED4E       pt_273
3ED58       pt_307
3ED62       pt_310
3ED6C       pt_355
3ED76       pt_358
3ED80       pt_337
3ED8A       pt_785
3ED94       pt_1041
3ED9E       pt_1042
3EDA8       pt_1092
3EDB2       pt_1105
3EDBC       pt_1106
3EDC6       pt_1296
3EDD0       pt_1297
3EDDA       pt_1360
3EDE4       pt_1552
3EDEE       pt_1553
3EDF8       pt_1616
3EE02       pt_2129
3EE0C       pt_2145
3EE16       pt_2146
3EE20       pt_2149
3EE2A       pt_2577
3EE34       pt_2578
3EE3E       pt_2581
3EE48       pt_2586
3EE52       pt_2593
3EE5C       pt_2594
3EE66       pt_2602
3EE70       pt_2641
3EE7A       pt_2657
3EE84       pt_2658
3EE8E       pt_2661
3EE98       pt_2721
3EEA2       pt_2722
3EEAC       pt_2730
3EEB6       pt_-1
3EEC0       real_137
3EED5       real_136
3EEEA       real_31
3EEFF       real_2.71828182846
3EF14       real_10
3EF29       real_180
3EF3E       real_360
3EF53       string_"YES"
3EF63       string_"NO"
3EF71       string_"Purge?"
3EF87       string_"Out of Memory"
3EFAB       string_"Stack"
3EFBF       string_"UNDO Stack"
3EFDD       string_"Command Stack"
3F001       string_"Last Arguments"
3F027       string_"System Object"
3F04B       string_"]"
3F057       string_"["
3F065       string_"|"
3F071       string_"{"
3F07D       string_"}"
3F089       string_"#"
3F095       string_"»"
3F0A1       string_"«"
3F0AD       string_"  "
3F0C1       string_NEWLINE
3F0D4       string_"der"
3F0E4       string_"IFTE"
3F0F6       string_"NOT"
3F106       string_" NOT "
3F118       string_"AND"
3F128       string_" AND "
3F13C       string_"XOR"
3F14C       string_" XOR "
3F160       string_"OR"
3F16E       string_" OR "
3F180       string_" "
3F18C       string_"LEVEL "
3F1A2       string_"UNKNOWN"
3F1BA       string_"""
3F1C6       string_"'"
3F1D2       string_","
3F1DE       string_"."
3F1EA       string_"("
3F1F6       string_")"
3F202       string_"^"
3F20E       string_"*"
3F21A       string_"/"
3F226       string_"+"
3F232       string_"-"
3F23E       string_"="
3F24A       string_"=="
3F258       string_"<"
3F264       string_">"
3F270       string_"√X"
3F27C       string_d/dx
3F288       string_"≤"
3F294       string_"≥"
3F2A0       string_"≠"
3F2AC       string_"1"
3F2B8       string_"2"
3F2C4       string_"3"
3F2D0       string_"4"
3F2DC       string_"5"
3F2E8       string_"6"
3F2F4       string_"7"
3F300       string_"8"
3F30C       string_"9"
3F318       CATALOG_key
3F5B0       DISP_4_semimenu
3F5B5       pt_23
3F5FB       labelall_1string2string3string4string5string6string
3F6F8       string_" USAGE: "
3F7AD       pt_39
3F82C       pt_20
3F84F       pt_21
3F863       pt_22
3F92B       pt_23
3F9C1       pt_29
3FC75       pt_260
3FF8B       string_" Version 2BB"
3FFAD       string_"Copyright HP 1986, 1987"
```

Wer jetzt also Erfahrungen mit einzelnen Funktionen oder Einsprungadressen macht sollte sie im Interesse der anderen Clubmitglieder so schnell wie möglich veröffentlichen, damit nicht allzu viele Erfahrungen doppelt gemacht werden müssen.

Wer tippt schon gerne seine Programme nach einem Rechnerabsturz komplett wieder ein, weil er etwas ausprobiert hat, was schon einem anderen vor ihm eine schlaflose Nacht bereitet hatte.

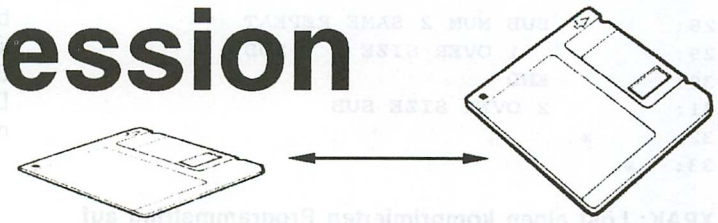
Eine nähere Erläuterung der Funktionen wurde vom Author mit dem Hinweis abgelehnt, er könne Ärger mit Hewlett Packard bekommen, was nicht in seinem Interesse liegen würde.

Ich habe so gut wie möglich versucht die Zeichen auf der Tastatur zu verwenden (Integral oder Wurzel etc.), damit die Assoziation möglichst leicht fällt.

Quelle:
Eric Toonen
Djept-Zuid 6
5502 PR Veldhofen
THE NETHERLANDS
Überarbeitung:
mm

Datenkompression

für den HP28/48



Das hier vorgestellte Programmpaket macht zum großen Teil Schluß mit dem Datenmüll des Rechners, indem nicht auf Variable, sondern auf überflüssige Leerbits verzichtet wird. Dies bedeutet, daß der Rechner alle Objekte, leider keine kompletten Verzeichnisse mit Unterverzeichnissen, komprimiert und auf unter 50% des ursprünglichen Speicherbedarf drückt. Dies behauptet zumindestens der Autor, Bob Peraino, George Mason University, USA).
Bei meinem Rechner (HP48) waren es zwar nur ca. 63% des ursprünglichen Platzbedarfs, aber das ist ja auch schon etwas...

Das vorgestellte Programm komprimiert sehr langsam (ca. 3kByte Programme in 4-5 Minuten), das Dekomprimieren geht dagegen relativ schnell (unter einer Minute), dafür aber sehr sicher. So gut wie alle Anwenderfehleingaben werden abgefangen und führen nicht zum Absturz, so daß die vorher abgespeicherten Objekte alle wieder zur Verfügung stehen.

Das Paket unterteilt sich in drei Teile: PAK/XPAK für Programme, MASH/XMASH für reele und komplexe Matrizen und einem "Serverpaket" für richtiges Aufrufen von Hand oder per Programm. Ich habe an den geeigneten Stellen auf eine mögliche Programmausführung hingewiesen.

Dieses Paket kann, je nach Anwender, entsprechend gestrafft werden, indem einzelne Programmteile weggelassen werden. Wichtig ist, daß dann die Variable "arck" für das "Serverpaket" entsprechend geändert werden muß. Wer also niemals Matrizen benutzt, der kann in "arck" das 4;5;15; und 16 Objekt durch einen Leerstring ersetzen.

Insgesamt ist der Platzbedarf für das Programmpaket ca. 3kByte, die sich aber sehr schnell amortisieren. Dadurch, daß die Komprimierung in der Regel nur einmal stattfindet, kann man sicherlich ganz gut mit diesem Programm leben, ich selbst habe jedenfalls schon lange keinen so aufgeräumten Rechner mehr gesehen.
Als Verzeichnisse benutze ich nur noch ein Verzeichnis für SYSEVAL-Experimente, ein Archiv-Verzeichnis als Unterverzeichnis einer TOOLBOX, in der alle nicht komprimierten Programme stehen, die für alle anderen Programme Hilfsdienste leisten (Anzeigeroutinen, Stringroutinen etc.) und ein TEST-Verzeichnis, in dem bei Bedarf die Anwendungen laufen bzw. ausgetestet werden.
Dadurch entfällt das lästige Umschalten in den Verzeichnissen usw. Die Arbeit wird einfach übersichtlicher. Dieses Verzeichnis ist ebenso ein Unterverzeichnis der TOOLBOX.

Zur Anwendung der Programme:
In Stack 2: sollte grundsätzlich das zu behandelnde Objekt stehen oder eine Liste derselben; in Stack 1: steht dann der Speichername des Archivfiles.
Für Matrixumformungen reicht das Objekt in Stack 1:.
Der Rest ist aus den ausführlich kommentierten Programmlistings zu ersehen.
Vielleicht schreibt ja ein anderer eine schnellere Version und/oder eine, die auch Verzeichnisse richtig bzw. komplett behandelt, das wäre genial.

PAK: Komprimiert ein Programm-Objekt

```
 1:  « →STR {" "} 10 CHR + → obj eoo          String bilden und Variable definieren
 2:    « "" DUP 1 obj SIZE                    Schleifenlänge für das Programm bestimmen
 3:       FOR i                               Beginn der Hauptschleife
 4:         i 3 ROLLD obj i i SUB eoo OVER    Ende der Objekte bestimmen
 5:         IF POS DUP THEN                   Wenn Ende, ...
 6:           IF 2 SAME THEN                  Wenn Ende ein NEWLINE-Zeichen ist ...
 7:             4 ROLL                        entferne es für andere Objekte
 8:             DO 1 + UNTIL                  innere Zählschleife
 9:               obj OVER DUP SUB " " ≠      letztes NEWLINE-Zeichen
10:             END                           Ende der inneren Zählschleife
11:             1 - 4 ROLLD                   Stelle Objekt wieder her
12:           END                             Ende der IF-Schleife
13:           DROP dict OVER                  Suche nach einem Gegenstück in "dict"
14:           IF POS DUP THEN                 Wenn in "dict" ...
15:             CHR SWAP DROP +               dann füge es in den zu bildenden String ein
16:           ELSE                            wenn nicht,
17:             DROP DUP SIZE 128 + CHR       setze Zeichen für "unbekannt",
18:             SWAP + +                      und füge dies entsprechend ein.
19:           END                             Ende der IF-Schleife
20:           ""                              Einen Leerstring für das nächste Objekt bilden
21:         ELSE                              oder noch keine Ende (korrespondiert mit der Zeile 5)
22:           DROP +                          füge ein Zeichen zu Objekt
23:         END                               Ende der IF-Schleife von Zeile 5
24:         3 ROLL i - 1 +                    Berechne die Größe der Schrittweite
25:       STEP                                Wiederhole Schleife FOR i ...
26:       DROP                                Lösche den letzten Leerstring
27:       WHILE DUP DUP SIZE DUP              Da letztes Zeichen eines Objektes "»" nicht benötigt wird
```

```
28:         SUB NUM 2 SAME REPEAT
29:           1 OVER SIZE 1 - SUB
30:         END
31:         2 OVER SIZE SUB
32:       »
33:   »
```

benötigt wird ...
lösche es.
Ende der WHILE-Schleife
Das erste Zeichen eines Objekt "«" wird ebenfalls nicht mehr benötigt.

XPAK: Löst einen komprimierten Programmstring auf

```
 1:   « → ob
 2:       « "«" 1 ob SIZE
 3:         FOR i
 4:           " " + ob i DUP SUB NUM DUP
 5:           IF 127 > THEN
 6:             ob SWAP 128 - i 1 +
 7:             SWAP DUP 1 + 5 ROLLD
 8:             i + SUB + SWAP
 9:           ELSE
10:             dict SWAP GET + 1
11:           END
12:         STEP
13:         STR →
14:       »
15:   »
```

nehme Objekt vom Stack
bestimme die Schleifengröße
Start
füge Stringzeichen zu, hole das nächste Byte
wenn höchstes Bit (Bit 7) gesetzt ist, dann ist es ein String
lösche höchstes Bit und zähle den Index
räume Wert für neue Schleife aus dem Weg
dekomprimiere den String
oder, wenn höchstes Bit nicht gesetzt ist,
dann handelt es sich um einen Befehl aus "dict"
Ende der IF-Schleife
wiederhole Schleife
löse den String wieder auf, dadurch dekomprimierte Objekte wieder auf den Stack

dict
{ "«" "»" . . .

"Wörterbuch" für PAK/XPAK
maximal 127 Befehle aus den am häufigsten vorkommenden Programmbefehlen

Die Vorschlagsliste setzt sich wie folgt zusammen:

```
{ "«" "»" "+" "DUP" "STO" "END" "SWAP" "→" "THEN" "IF" "NEXT" "FOR"
  "GET" "-" "DROP" "SIZE" " →STR" "*" "SUB" "{" "}" "PURGE" "SAME"
  "ELSE" "PATH" "RCL" "STR →" "CHR" "TYPE" "NUM" "PUT" "→LIST"
  "EVAL" "/" "^" "R→B" "DISP" "POS" "MENU" "NOT" ">" "DO" "UNTIL"
  "START" "HOME" "PIXEL" "[" "]" "CLLCD" "STO+" "R→C" "RDM" "B →R"
  "SQ" "ROT" "PICK" "SF" "REPEAT" "WHILE" "ROLL" "ROLLD" "OR" "IP" "≥"
  "MOD" "DROP2" "AND" "FS?" "#" "OVER" "IFERR" "KEY" "LIST→" "STOF"
  "CRDIR" "VARS" "FC?" "PMIN" "PMAX" "RAND" "SIN" "COS" "C →R" "FP"
  "<" "≤" "DGTIZ" "DEPTH" "DRAX" "BEEP" "ROLL2" "CLEAR" "RCLF" "HALT"
  "STO-" "CF" "≠" "INV" "SYSEVAL" "STWS" "XOR" }
```

Diese Liste ist laut Autor nach Schnelligkeit und Häufigkeit des Vorkommens optimiert, es macht aber garnichts, wenn nur die beiden ersten Zeichen "«" und "»" in "dict" stehen.

Die Anwesenheit dieser beiden Zeichen ist aber absolute Pflicht, ansonsten gibt es CHAOS !!

Und wie gesagt, es sind nur maximal 127 Befehle zulässig, dies ist durch den Verschlüsselungsalgorithmus so vorgegeben.

MASH: Komprimiert eine Matrix, reel oder komplex

```
 1:   « 1 OVER TYPE
 2:       IF 4 SAME THEN
 3:         SF
 4:       ELSE
 5:         CF
 6:       END
 7:       ARRY → DUP →STR 3 OVER SIZE 2 - SUB
 8:       "}" + 1 FS? CHR SWAP + DEPTH ROLLD
 9:       LIST →
10:       IF 2 SAME THEN
11:         *
12:       END
13:       1
14:       FOR j
15:         j ROLL →STR
16:         IF 1 FS? THEN
17:           2 OVER SIZE 1 - SUB
18:         END
19:         DUP SIZE DUP CHR 3 ROLLD → ob sz
20:         « 1 sz
```

bestimme Typ, die Flagnummer (siehe auch letzte Zeile)
ist es eine komplexe Matrix ?
wenn ja, dann setze Flag 1
wenn nicht,
dann lösche Flag 1
Ende der IF-Schleife
löse den Array auf und hole seine Größe
Achtung: Listenklammer rechts, Matrixtyp
Dimensionsgröße auflösen
wenn 2-dimensional
bestimme TOTAL# der Elemente
Ende der IF-Schleife
füge Schleifenende an
Schleife für die Elemente
bringe nächstes Element an den Anfang
wenn es eine komplexe Matrix war
dann entferne die Klammern
Ende der IF-Schleife
Größe der Objekte, ehemalige Größe Obj
Schleife beginnen

```
21:          FOR i
22:            mlex ob i i SUB POS 16 *
23:            IF sz i > THEN
24:              mlex ob i 1 + DUP SUB
25:              POS +
26:            END
27:            CHR +
28:          2 STEP
29:        »
30:      DEPTH ROLL SWAP + DEPTH ROLLD
31:    -1 STEP
32:    DEPTH ROLL
33:  »
```

Schleifenzähler für Elemente
finde Zeichen-Bit, rotiere eins nach links
Größe könnte ungerade sein
so nimm nächstes Zeichen und setze es ein
das Low-Byte des nächsten Zeichens
Ende der Schleife
verändere Zeichen, füge es in den Puffer ein
Ende der Matrixelemente
füge Puffer an den Endstring an
Ende der Matrix
Für den Fall, daß die Matrix innerhalb einer anderen Anwendung war, hole sie in der ersten Zeile

XMASH: Dekomprimiert ein Matrixobjekt

```
 1:  « DUP 1 1 SUB NUM "{" "" IFTE → ob t
 2:    « ob "}" POS 1 + ob SIZE
 3:      FOR i
 4:        t i 1 + ob i i SUB NUM 2 /
 5:        DUP 4 ROLLD i + .5 +
 6:        FOR j
 7:          mlex ob j IP DUP SUB NUM
 8:          DUP 16 / IP
 9:          IF j FP .5 SAME THEN
10:            16 * -
11:          ELSE
12:            SWAP DROP
13:          END
14:          DUP SUB +
15:        .5 STEP
16:        STR→ SWAP DUP
17:        IF FP .5 SAME THEN
18:          .5 +
19:        END
20:        1 +
21:      STEP
22:      "{" ob 2 ob "}" POS SUB +
23:      STR→ →ARRY
24:    »
25:  »
```

wenn Matrix komplex ist, dann Klammern berücksichtigen
berechne die Größe des komprimierten Strings
Schleife, um alle Bytes zusammen zu setzen
nehme Größe des Elements, und ...
lege eine Kopie auf den Stack
Schleife durch die Elemente
nehme nächstes Byte aus dem String
und hole oberes Nibble des Zeichens
wenn wir gerade beim Lower-Nibble sind,
dann hole es
ansonsten
streiche unbenötigte Kopie
Ende der Schleife
hole Charakterzeichen für die Zahl, füge es an
hole nächstes Zeichen aus der Matrix
Konvertiere Matrix zu #, hole Ursprungsgröße
wenn Größe ungerade
dann runde auf
Ende der Schleife
um erstes Byte des nächsten Elements zu bestimmen
wiederhole Schleife und hole Größe des nächsten Elements
stelle Dimensionsgröße wieder her
und forme Ursprungsmatrix

mlex

```
"0123456789,.E-"
```

Konstante für MASH/XMASH
wird in beiden Programmen benötigt

ARC: Serverroutine für mehrere Anwendungen

```
 1:  « → ls af
 2:    « 31 CF af
 3:      IFERR RCL THEN
 4:        {}
 5:      END
 6:      1 ls SIZE
 7:      FOR i
 8:        ls i GET DUP "ARCing "
 9:        OVER →STR + 1 DISP
10:        IFERR RCL THEN
11:          DROP
12:        ELSE
13:          DUP TYPE → t
14:          « arck t 1 + GET
15:            STR→ → on ob
16:            « DUP
17:              IF on POS DUP THEN
18:                SWAP OVER 1 - ob PUT
19:                SWAP 1 + t PUT
20:              ELSE
21:                DROP ob + on + t +
22:              END
23:            »
```

Hole Fileliste und Archivfile-Namen
für HP28; für HP48: -55 CF, ≠ LASTARG
wenn noch kein Archivname existiert,
dann starte einen neuen
Ende der Schleife
hole die Anzahl der verschlüsselten Objekte
Schleife für Elemente
hole File-Namen
zeige aktualisiertes Display
wenn File nicht existiert,
dann lösche Stackzeile
oder
hole den Typ des Objekts
hole Befehlsroutine aus "arck"
Befehl und Objekt in die Variable
Objekt
wenn Objekt schon existiert,
dann ersetze altes durch neues
ergänze Typ des Objektes
oder
füge neues hinzu
Ende der Schleife

```
24:            »
25:          END
26:        NEXT
27:        af STO CLMF
28:     »
29:  »
```

Ende der Schleife von Zeile 10:
Schleifenwiederholung
HP28-Version; HP48-Version: `af STO TEXT`

Hinweis: in der Zeile 27: kann nach "af" zuerst das Archiv-Verzeichnis aufgerufen werden, um die Abspeicherung ebenfalls zu automatisieren. Auch empfiehlt es sich, in Zeile 1: ein `RCLF → FLAGS` vorzuschalten, um die ursprüngliche Flagstellung der anderen Programme zu sichern.
Nach `af ...` (ARCHIV) `STO` könnte dann `FLAGS STOF` eingesetzt werden.

XARC: Holt Objekte aus dem ARCHIV-File

```
 1:  « ' ls af
 2:     « 1 ls SIZE
 3:        FOR i
 4:          ls i GET ' on
 5:          « "XARCing " on →STR
 6:             + 1 DISP af RCL DUP
 7:             DUP on
 8:             IF POS DUP THEN
 9:               SWAP OVER 1 - GET
10:               3 ROLLD 1 + GET
11:               12 + arck SWAP GET
12:               STR ' on STO
13:             ELSE
14:               3 DROPN
15:             END
16:          »
17:        NEXT
18:        CLMF
19:     »
20:  »
```

speichere Name der Liste und Speichername
Anzahl der Filenamen
Schleife
hole Objektname
aktualisiere Display
hole ARCHIV-File
und Objektnamen
wenn Name ein ARCHIV-File,
dann hole das Objekt
hole Objekttyp
hole entsprechenden Befehl für den Objekttyp
Typ, Ausführungsroutine, speichere OUTPUT
oder
wenn nicht gefunden, streiche Objekt
Ende der Schleife

wiederhole Schleife
HP28-Version; HP48-Version: `TEXT`

ARCL: Inhaltsliste eines ARCHIV-Files

```
1:  « RCL ' ls
2:     « {} 2 ls SIZE
3:        FOR i
4:          ls i GET +
5:        3 STEP
6:     »
7:  »
```

hole ARCHIV-File
Anzahl der Objekte
Schleife
füge Objektnamen zur Liste
hole nächstes Objekt
Hinweis für Programme: `NAME DUP ARCL SWAP XARC...`

arck: Konstante für ARC/XARC

```
{"" "" "" "MASH" "MASH" "1 →LIST" "→STR" "" "PAK" "→STR" "" "" "" "" "XMASH" "XMASH" ""
"STR '" "" "XPAK" "STR '" ""}
```

Falls XARC ebenfalls "automatisch" die Objekte oder Programme in ein bestimmtes Verzeichnis kopieren soll, so ist in Zeile 12 nach `STR '` das Verzeichnis einzufügen und ebenfalls in Zeile 12 nach `STO` wieder das ARCHIV-Verzeichnis aufzurufen.

Georg Hoppen
Hubertusring 5
4512 Wallenhorst

Tippfehler in "Innereien des HP48SX"

aus PRISMA 5-6/90

Beim Eintippen des oben genannten Artikels sind mir bedauerlicherweise zwei Tippfehler unterlaufen, woraufhin die Routine "PEEK" nicht funktionieren kann.

```
« RCWS SWAP 64 STWS #0h OR SWAP STWS
"ABCDEFGHIJKLMNOPQRSTUVWXYZ" ...
```

Hier fehlte im Alphabet das T. Wer in der Schule aufgepaßt hätte, dem müßte dies aufgefallen sein.

Genützt hätte ihm dies aber leider auch nicht, da in der Zahlenkette auf Seite 38 noch ein Zahlendreher drin war:

```
13210314313016 9146...
```

Die Zahlen 6 und 9 waren im Artikel vertauscht worden, ich hoffe damit nicht zu viele Abstürze provoziert zu haben. Sorry, abends um 12 verschwimmen solche Zahlenkolonnen schon vor den Augen. mm

28S: Programmhinweise

von Ralf Pfeifer

In seinem Artikel über C-Programme (89.6.4) schreibt Frank Rieg, daß es für jedes Programm eine optimal geeignete Sprache gibt. Da sich das Betriebssystem im HP-28 allerdings nicht wechseln läßt, möchte ich hier einige Strategien zum Ersatz von GOSUB/XEQ innerhalb eines Programms und für die in BASIC bekannte ON...GOTO Anweisung vorstellen.

Mit den IF/THEN-Tests lassen sich nur max. 2 Verzweigungen realisieren, wobei eine 1 das Programm nach THEN, eine 0 nach ELSE weiterlaufen läßt. Im Amiga-BASIC verlangt der Computer zwischen ON und GOTO/GOSUB eine ganze Zahl ab 1 aufwärts. Hinter GOTO/GOSUB stehen dann (beliebig viele) Zeilennummern, zu denen verzweigt werden kann. Sollte es zu der ganzen Zahl keine Adresse geben, arbeitet das Programm einfach in der nächsten Zeile weiter. Auf dem HP-28 läßt sich diese Struktur mit Listen verwirklichen.

Das erste Beispiel **NGL** benötigt als Unterprogramme GL2, GL3 und GL4 aus PRISMA 89.4.35; NGL erwartet in Ebene 1 einen Vektor, der alle n+1 Koeffizienten eines Polynoms n-ten (n bis 4) Grades erwartet. NGL stellt dann den Grad des Polynoms durch die Anzahl der Elemente im Vektor fest, und teilt den Vektor durch das erste Element (bringt das Polynom auf Normalform), von der Größe des Vektors wird 1 abgezogen und in der lokalen Variablen n gespeichert. Die Liste in Zeilen 3-4 enthält dann die 4 Verzweigungsmöglichkeiten (was in BASIC nach GOTO/GOSUB kommt), an erster Stelle die Funktion NEG aus dem REAL-Menue (Listen können auch einzelne Standardfunktionen enthalten!) und dann die Namen der eben genannten Unterprogramme. "n GET" bringt dann ein Objekt aus der Liste in Ebene 1, dort angekommen werden da aber weder NEG noch die Namen von selbst aktiv, deshalb noch ein EVAL. Der Rest verpackt dann die Lösungen wieder in ein Array (hier genügt eine komplexe Lösung, und andere reelle Lösungen werden ebenfalls in die komplexe Darstellung umgewandelt).

TYP ist ein weiteres Beispiel für die ON..GOTO-Adaption: Das Programm bestimmt mit TYPE die Art eines Objekts und gibt ihn dann im Klartext aus. An Position 1 der Liste (Zeilen 1-20) steht der Klartext zum TYPE 0, an Position 2 der zu TYPE 1 usw. Für die Objekttypen 3, 4 und 8 (Positionen 4,5,9) liegen keine Strings, sondern Programme vor. 3 und 4 bedeuten Arrays mit reellen oder komplexen Zahlen. Die kleinen (Position 4,5) Programme machen sich hier die Mühe zu unterscheiden, ob eine Matrix oder ein Vektor vorliegt, und beim Typ 8 prüft das kleine Programm, ob es sich um ein Programm (also « ») oder eine Funktion handelt.

Solche Funktionen (wie NEG im Beispiel NGL) kann man so in den Stack bringen: Liste beginnen, und Name der Funktion (z.B. SIN) eintippen, ENTER 1 GET - fertig.

Doch zurück zu TYP, wenn TYPE das Objekt erkannt hat, holt es das zugehörige Objekt aus der Liste. Programme werden mit EVAL gestartet, Strings macht EVAL nichts aus. Der Rest ist Arbeit für's LCD. Für die Auswahl der richtigen Position in der Liste erhielten die beiden vorgestellten Programme auf "natürliche" Weise ganze Zahlen.

Wie man in anderen Fällen zu solchen Zahlen kommt, zeigt **EX0**, welches Tastatur abfragt, aber nur die Tasten A, B, ENTER und BACK zuläßt, die übrigen aber ignoriert. Die innere (zweite) DO/UNTIL-Schleife wartet auf einen Tastendruck, der dank Key einen String erzeugt. Falls dieser String in der Liste (Zeilen 2-3) nicht enthalten ist, liefert POS eine 0, und die Schleife läuft weiter, andernfalls kommt eine Zahl zwischen 1 und 4 heraus.

Ein etwas anderer Weg, mehrfach zu verzweigen, bietet die Rekursion. Das Programm SORT in PRISMA 89.6.38 kann nur Listen sortieren, obwohl es durchaus sinnvoll sein kann, auch den Stack oder ein Array (Matrix oder Vektor) zu ordnen. Die beste Lösung wäre wohl, ein Programm zu schreiben, welches entweder den Stack sortiert (das erledigen die Zeilen 2-11 in 89.6.38), oder bei Listen Arrays diese auflöst, die Elemente im Stack ordnet und dann wieder eine Liste/Array speichert.

Das neue **SORT** möchte ich deshalb hier vorstellen, weil es die kürzeste von mir erstellte Routine war, und weil es rekursiv programmiert ist.

SORT erlaubt folgende Eingaben:

a) Stack sortieren. In Ebene 1 steht die Anzahl (n) der zu sortierenden Ebenen (2 bis n+1). Nach Programmende ist n verschwunden, das kleinste Element befindet sich in Ebene n, das größte in Ebene 1. n entspricht also der Form, wie sie auch von DROPN, ROLL oder →LIST verlangt würde. Zum Sortieren sind Objekte vom Typ "Real", "String" und "Binary" (nicht vermischt!) zugelassen.

b) Liste sortieren. Die Liste steht in Ebene 1, und ist nach Programmschluß geordnet, kleinstes Element links. Zugelassener Inhalt wie a).

c) Array (Matrix oder Vektor) sortieren. Das Objekt darf nur reelle Zahlen enthalten und wird in Ebene 1 erwartet.

```
    NGL
001 « DUP 1 GET / ARRY→
    LIST→ - → n
003   « ( NEG GL2 GL3
    GL4 ) n GET EVAL n
005   » →ARRY SWAP DROP
    » 95,5 Bytes

1:    [ 1 -1 -21 1 20 ]
NGL
1:        [ -4 -1 5 1 ]

    TYP
001 « { "REAL NUMBER"
    "COMPLEX NUMBER"
003 "STRING"
      « "REAL " OVER
005 SIZE SIZE 1 SAME
    "VECTOR" "MATRIX"
007 IFTE +
      »
009   « "COMPLEX " OVER
    SIZE SIZE 1 SAME
011 "VECTOR" "MATRIX"
    IFTE +
013   » "LIST"
    "GLOBAL NAME"
015 "LOCAL NAME"
      « DUP →STR 1 1 SUB
017 "«" SAME "PROGRAM"
    "FUNCTION" IFTE
019   » "ALGEBRAIC"
    "BINARY NUMBER" }
021 OVER TYPE 1 + GET
    EVAL "TYPE " ROT
023 TYPE →STR + " : " +
    SWAP + 1 DISP
025 » 376,5 Bytes

1:          [[ 1 2 3 ]]
TYP
TYPE 3 : REAL MATRIX

    EX0
001 «
      DO { "A" "B"
003 "ENTER" "BACK" }
        DO KEY
005     UNTIL
        END
007   UNTIL POS
      END DUP
009 » 76 3_ es

    SORT
001 « DUP
      IF TYPE
003   THEN LAST OVER
    SIZE → n
005     « 3
          IF SAME
007       THEN ARRY→
    LIST→ DUP DUP DROPN
009 « SORT n →ARRY
        ELSE LIST→
011 SORT n →LIST
        END
013     »
      ELSE 2
015     FOR a 2 a
          START DUP2
017         IF >
            THEN SWAP
019         END a ROLLD
        NEXT LAST
021 ROLLD -1
      STEP
023   END
    » 187 Bytes
```

```
    REV
001 « DUP
      IF TYPE
003   THEN SIZE LAST DUP
    TYPE 3
005     IF SAME
        THEN ARRY→ LIST→
007 DUP DUP DROPN * 1 +
    REV →ARRY
009     ELSE + LIST→ REV
    →LIST
011     END
      ELSE 2 SWAP
013     FOR n n ROLL
        NEXT
015   END
    » 142 Bytes


1:          [ 1 5 0 4 -2 ]
SORT
1:          [ -2 0 1 4 5 ]
REV
1:          [ 5 4 1 0 -2 ]

    PRIM
001 « IP ABS "" SWAP
      « ROT DUP SIZE "*"
003 "'" IFTE + OVER →STR
    + ROT ROT 0 4 ROLLD
005     DO SWAP OVER /
    SWAP 4 ROLL 1 + 4
007 ROLLD
        UNTIL DUP2 MOD
009     END ROT 4 PICK 1
    > "^" 6 ROLL →STR +
011 "" IFTE + DUP 1 DISP
    ROT ROT
013   » → d
      « 2 DUP2 < DROPN
015 DUP2 MOD NOT d IFT
    DROP 3 DUP2 MOD NOT
017 d IFT DROP 5 DUP2
    MOD NOT d IFT SIGN
019     DO 6 + DUP2 MOD
    NOT d IFT 4 + DUP2
021 MOD NOT d IFT 2 +
    DUP2 MOD NOT d IFT 4
023 + DUP2 MOD NOT d IFT
    2 + DUP2 MOD NOT d
025 IFT 4 + DUP2 MOD NOT
    d IFT 6 + DUP2 MOD
027 NOT d IFT 2 + DUP2
    MOD NOT d IFT
029     UNTIL DUP2 SQ <
        END DROP DUP DUP
031 1 ≠ d IFT DROP2
      » CLMF STR→
033 » 446,5 Bytes

12 FACT PRIM
1:  '2^10*3^5*5^2*7*11'

    EX1
« « n IDN »
 → n p « p EVAL » »

    EX2
« → n « « n IDN »
    → p « p EVAL » » »

    DOP
001 « 1 - 137 * LCD→ 1
    ROT SUB LAST + DUP
003 136 + SUB SWAP 1 69
      FOR a OVER a DUP
005 SUB DUP + +
      NEXT SIZE LAST
007 OVER 1 - 1 SWAP SUB
    LCD→ ROT 548 SUB +
009 +LCD DROP
    » 158,5 Bytes

4 DOP
→ARRY ARRY→ PUT
```

Nach der Bearbeitung ist das Element 1 bzw. 1,1 das Kleinste.

Intern läuft das Programm SORT so ab: Zunächst prüft es, ob sich in Ebene 1 eine reelle Zahl befindet. Falls ja, wird der Stack sortiert, und das Programm beendet. Falls nein, prüft das Programm, ob entweder ein Array oder eine Liste vorliegt, und schiebt deren Elemente dann auf den Stack. Die Anzahl dieser Elemente kommt in Ebene 1 und damit liegt der Fall "Stack sortieren" vor, so daß sich jetzt SORT selbst aufruft, also in die erste (und einzige) Rekursionsebene einsteigt.

REV (REVerse) kehrt die Ordnung der Elemente in Stack, Arrays oder Listen um, wobei das Programmgerüst von SORT fast unverändert übernommen wurde, so daß sich auch in der Bedienung nur folgende Unterschiede ergeben: Stack oder Listen dürfen sogar beliebige Objekte enthalten. Auch Arrays werden umgekehrt, allerdings in einer Weise, die mit TRN nicht zu erreichen ist.

Eine Methode, Unterprogramme ins Hauptprogramm zu holen, bieten lokale Variable, die können nämlich auch Programme speichern, was ich am Programm PRIM (PRISMA 88.5.44) zeigen möchte. Das Programm versucht, die eingegebene Zahl durch andere zu teilen. Gelingt das ohne Divisionsrest, ist ein Teiler gefunden, den das Unterprogramm DV herausfindet. DV ist Programm in einer globalen Variablen, und taucht DV in irgendeinem anderen Programm auf, so verzweigt dieses sofort ins Unterprogramm DV. Um das zu verhindern wurde stattdessen 'DV' (also mit ') programmiert. Die hier vorgestellte Version von PRIM enthält DV als eigenes abgeschlossenes Programm in den Zeilen 2-12, welches in die lokale Variable d (Zeile 13) gespeichert wird. Der Aufruf der lokalen Variablen d führt das Unterprogramm aber nicht aus, sondern wirkt so wie ein 'DV' RCL, so daß noch ein EVAL folgen müßte, um das Programm in Ebene 1 zu aktivieren. der folgende IFT-Befehl wirkt dann (je nach Testergebnis) wie ein EVAL oder DROP.

Das hier abgedruckte PRIM funktioniert etwas anders als das in 88.5.44, es arbeitet jetzt mehr wie eine Standardfunktion. zunächst erwartet PRIM in Ebene 1 eine positive Ganzzahl, und gibt entweder die Zahl zurück (TYPE 0, falls prim) oder liefert die Zerlegung als algebraisches Objekt (TYPE 9). Eine weitere Testmöglichkeit ist: DUP PRIM SAME, 1 bedeutet prim.

Lokale Variable halten allerdings noch andere Überraschungen bereit. Die Experimentierprogramme **EX1** und **EX2** haben zwar keinerlei Sinn, dafür sind sie pädagogisch wertlos. Bei EX1 hat sich der Programmierer folgendes gedacht: EX1 nimmt eine Zahl (z.B. 5) vom Stack und erzeugt mit IDN eine Einheitsmatrix entsprechender Größe. Der Ablauf soll so aussehen: Zunächst legt EX1 das Unterprogramm « n IDN » im Stack ab. Dann werden zwei lokale Variable erzeugt, n mit der Zahl (hier 5), die EX1 braucht, und p mit dem Unterprogramm. p EVAL soll nun das Unterprogramm ausführen, welches n (also 5) holt, doch IDN unterbricht hier mit einem "Undefined Name" Error, und 'n' steht in Ebene 1, denn der HP-28 hat von alledem eine ganz andere Meinung. Nach dem Start von EX1 liest er zuerst das Unterprogramm, *und findet n* zu einem Zeitpunkt an dem die lokale Variable n noch garnicht definiert ist. Mangels höherer Einsicht hält er dieses erste n also für eine globale Variable, und übersetzt sich sein Programm entsprechend.

Als nächstes definiert er die lokalen Variablen n und p, aber beim Versuch, das in p gespeicherte Unterprogramm auszuführen, stößt er wieder auf n. Da er dieses vorher schon als globale Variable einstufte, sucht er auch jetzt wieder danach - natürlich erfolglos, was einen Error provoziert. Noch wildere Ergebnisse erhält man bei der Wahl von i oder e statt n, denn diese gelten im HP-28 als numerische Konstanten. Das Verhalten des HP-28 wiederholt sich sinngemäß, wenn FOR lokale Variable erzeugt. Wie es richtig funktioniert zeigt EX2.

DOP erzeugt die doppelte Zeichenbreite auch in der Anzeige. Das Prinzip ist dasselbe wie beim Drucker: Jede Spalte eines Zeichens wird verdoppelt. Wie die Funktion DISP unterteilt DOP die Anzeige in 4 Zeilen. Jede Display-Zeile hat 137 Spalten von denen die ersten 68 verdoppelt, die 69. nur einfach angehängt, und die Spalten 70-137 vergessen werden.

Das Programm MKRY (MiniKRYpt) ist ein sehr kurzes und schnelles Verschlüsselungsprogramm, welches das Prinzip von Vigenére variiert: Statt den Buchstaben von Klartext und Schlüssel Zahlen zuzuordnen, die mit "+" verknüpft werden, nimmt dieses Programm die Funktion "XOR". Vorteil: Das Programm ist selbstinvers, d.h. dasselbe Programm, welches verschlüsselt, kann auch wieder entschlüsseln, es "toggelt" also (mit Hilfe des Schlüssels) zwischen Klartext und Code. Nachteil: Es sind alle 256 Zeichen zugelassen, und das Ergebnis kann auch aus diesen bestehen. Das Programm eignet sich also nur dazu, Strings im Rechner vorübergehend unlesbar zu machen, aber nicht zum Nachrichtenaustausch zwischen zwei Rechnern. MKRY erwartet zwei Strings, in Ebene 2 den Klartext oder Code und in Ebene 1 den Schlüssel.

Ein Wort zur Programmpflege: In PRISMA 89.4.31 habe ich das Programm DATE vorgestellt, und zugleich die Ungenauigkeit der eingebauten Uhr bemängelt. In Zeile 5 des DATE Programms (89.4.32) findet sich eine 29491200, die beim HP-28 zur Korrektur verwendet werden kann. Diese Zahl gibt an, wie oft die Uhr in jeder Stunde tickt (3600 Sekunden pro Stunde mal 8192 "ticks" pro Sekunde). Im Falle meines Rechners habe ich diesen Wert auf 29491075 verringert, und

erhalte seitdem über Monate hinweg sehr genaue Uhrzeiten. Dabei ist dieser Faktor zu erhöhen, falls die Uhr vorgeht, und unter den Normalwert 29491200 zu erniedrigen, falls die Uhr nachgeht. Dabei sollte man nicht kleinlich sein, denn ändert man diesen Normalwert um 1 bedeutet das nur 1 s Genauigkeitsveränderung in 10 Jahren. H. R. Wuttke erwähnt in PRISMA 89.6.16, daß die Feinjustierung der Uhr im HP-71 nicht sinnvoll ist. Obwohl der HP-28 neben seiner "Saturn"-CPU sicherlich die gleichen Bauteile verwendet, kann ich eine Justierung (auf die Sekunde genau) empfehlen.

Hier noch ein paar Programmschnipsel, also Lösungen für Probleme, die manchmal in Programmen auftauchen, und für die man eine elegante Lösung sucht.

Problem A: Ein Programm läuft, und es will vom Anwender mehrere Tastendrükke als Entscheidungshilfe haben. Für einen Tastendruck empfiehlt das HP-28 Handbuch die Abfrage ...DO UNTIL KEY END...; Für mehrere Tasten (z.B. 5) empfehle ich **A1**. Wird eine andere Anzahl als 5 gewünscht, ist diese im Programm entsprechend abzuändern.

A2 löst dasselbe Problem, zusätzlich wird jeder Tastendruck mit einem kurzen Pieps quittiert.

Während ein Programm oder eine Funktion abläuft, werden bis zu 15 Tastendrücke im Key-Buffer (Tastendruck-Speicher) vom Rechner festgehalten, bis entweder das Programm/Funktion anhält (die Tastendrücke werden ausgeführt) oder die Funktion KEY den Pufferspeicher Stück für Stück leert.

Soll der HP-28 nach Programmende versehentlich oder absichtlich betätigte

```
    A1
001 « 1 5
      START KEY
003     STEP
    » 50 Bytes

    A2
001 « 1 5
      START KEY
003     IF DUP
        THEN 1000 ,01
005 BEEP
        END
007     STEP
    » 68,5 Bytes

    A3
001 «
      WHILE KEY
003   REPEAT DROP
      END
005 » 30 Bytes

    B
001 « SIZE LIST→ DUP DUP
    DROPN *
003 » 32,5 Bytes

    C
001 « IM LAST DUP RE
    IFTE
003 » 30 Bytes

    MKRY
001 « OVER SIZE SWAP
      DO DUP +
003   UNTIL DUP2 SIZE <
      END 1 ROT SUB XOR
005 » 55 Bytes

2:   "HP-28S MKRY TEST"
1:                "CCD"
MKRY
1:   "▪▪iq{▪c▪▪▪▪d▪▪▪▪"

2:   "▪▪iq{▪c▪▪▪▪d▪▪▪▪"
1:                "CCD"
MKRY
1:   "HP-28S MKRY TEST"
```

Funktionen nicht ausführen, hängt man **A3** an um den Key-Buffer zu löschen.

Problem B: Ebene 1 enthält ein Objekt, das Type mit 3 (reelles Array) oder 4 (komplexes Array) kennzeichnet. Es könnte sich dabei also um einen Vektor (SIZE liefert Liste mit einem Element) oder um eine Matrix (SIZE liefert eine Liste mit zwei Elementen) handeln, Frage also, wieviele Zahlen enthält dieses Array? Die Lösung berechnet **B**.

Problem C hat man, wenn der HP-28 aus einer reellen Zahl wieder einmal eine komplexe gemacht hat, denn er behält diese auch dann bei, wenn der Imaginärteil 0 ist. Abhilfe schafft Schnipsel **C**.

Wer seine Dezimalzahlen mit Kommas bestückt (wer also im MODE-Menue RDX, bzw. 48 SF gewählt hat), kann dieses Komma auch in Namen verwenden, nicht jedoch den Punkt. Das kann man ändern, indem man vorübergehend (während der Eingabe bis zum ENTER) den Zustand von Flag 48 unkehrt. Ein solches Programm läßt sich dann später nur nach erneuter Änderung mit RCL oder VISIT untersuchen.

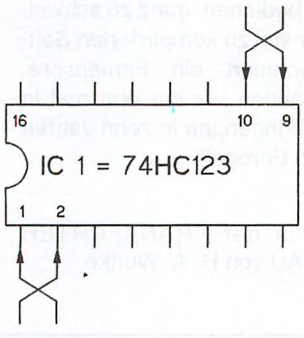
Hier noch eine Vorsichtsmeldung: SAME vergleicht Objekte in Ebene 1 und 2 bitweise, ist sehr schnell und ringt sich nur bei absoluter Gleichheit zu einer 1 durch. Dagegen prüft == die mathematische Gleichheit, was aber nur mit algebraischen Objekten vernünftig funktioniert. So ergibt '12/3 ' 4 == EVAL zwar 1 (= wahr), dummerweise hat man bei HP nicht daran gedacht, diese Funktion z.B. auf das Zusammenleben von reellen und komplexen Zahlen auszudehnen: (2.0) 2 == ergibt leider 0 (= falsch).

Ralf Pfeifer (116)

Nachtrag zum "Input-Output Board"

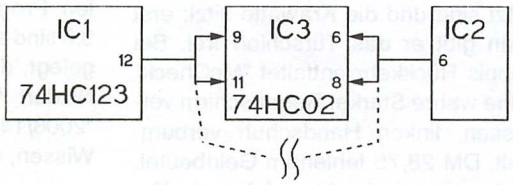
aus PRISMA 4/90

Im PRISMA 4/90 hatte Christoph Klug eine Schaltung veröffentlicht, mit der es möglich war, vom HP-IL Converter aus die Außenwelt anzusteuern.
Inzwischen kamen noch einige Änderungen bzw. Ergänzungen dazu, die ich jetzt auflisten möchte:
⇨



Anschluß Pin { 1, 2, 9, 10 } vertauschen.

Die nächste Änderung stelle ich auch besser in einer Zeichnung dar, damit man sie besser verstehen kann:
Die Verbindung zwischen IC1 Pin 12 zu IC3 Pin 8 unterbrechen; diese dann an den Pin 9 und an den Pin 11 von IC 3 legen.

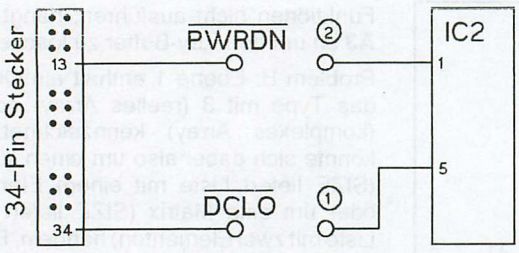


Die Verbindung zwischen IC2 Pin 6 auf IC3 Pin 6 und 8 bleibt dagegen bestehen!

⇨ Die Verbindung von IC2 Pin 3/4 auch auf Pin 12/13 von IC2 legen.

⇨ Steckerpin 6 {$\overline{\text{RDYI}}$} auf GND⊥ des 34-Pin Steckers.
Steckerpin 11 {DACI} auf GND⊥ des 34-Pin Steckers.

⇨ Den Steckerpin 13 {$\overline{\text{PWRDN}}$} auf IC2 Pin 1 mit Einschleifpunkten versehen.
Ebenso könnte der Steckerpin 34 {DCLO} auf IC2 Pin 5 mit einem Einschleifpunkt versehen werden, um später Systemerweiterungen anschließen zu können.
Das Bild siehe nächste Seite.



› Auf dem Layout fehlt der R6=10kΩ. Hier kann aber die Drahtbrücke zwischen R7=10kΩ und IC6 durch R6 ersetzt werden.

⇨ *Hinweis zum A/D-Wandler:*
Wird der 8-Bit A/D-Wandler benutzt, so muß IC12 = 74HC574 aus der Fassung herausgenommen werden. "IN2" ist als Port an dem 64-Pin Stecker nun nicht mehr verfügbar.

⇨ Alle IC's außer Operationsverstärker und Wandler sollten der HC-Logik Familie angehören, damit es keine Pegelanpassungsschwierigkeiten geben kann.

Christoph Klug
Leibnitzstraße 19
3200 Hildesheim

Glosse:

Immer elektronischer

von Gerhard Wittenberg

Auch in diesem Jahr schickt sich die "Cebit" an, eine Ausstellung der Rekorde zu werden: Noch mehr Aussteller, noch mehr Besucher, 295 Weltneuheiten (im Vorjahr 257), noch höhere Eintrittspreise - und Computer, die nicht nur besser, schneller, sondern auch menschlicher geworden sind. Für nahezu jede Berufsgruppe und jeden Typus hat die Computerindustrie etwas zu bieten - ob Mathematiker, Praktiker, Apotheker oder Astmathiker.

Zum Beispiel den Schusseligen. Dem "McCheck" von Appel mit hochmodernem 20MBDRAM, gezeigt in Halle 1, entgeht nichts: Will das eilige Familienoberhaupt frühmorgens aus dem Haus stürmen, wird er an alles erinnert, was noch fehlt - Führerschein, Kamm, Apfel in der Aktentasche usw. Gleichzeitig prüft "McCheck", ob die Schuhe geputzt sind und die Krawatte sitzt; erst dann gibt er das Türschloß frei. Bei Pappis Rückkehr entfaltet "McCheck" seine wahre Stärke: Regenschirm vergessen, linken Handschuh verbummelt, DM 28,75 fehlen im Geldbeutel. Infolgedessen druckt er folgende Befehlsliste: "Zurück ins Büro, Taschengeldbudget überzogen, Kopie an Hausfrau".

Ganz auf Zahnärzte zugeschnitten ist der "PriDent" von Hewlett Packard mit 256 Ein-/Ausgabeports. Davon schließt der Doktor freilich nur 2 an den Patienten an. "PriDent" diagnostiziert dann mit ungeahnter Präzision die Zahl der noch nicht gezogenen Zähne und führt eine Anamnese der bisherigen Behandlungen respektive Zahlungen der Kasse durch. Sollten die hochgerechneten Einnahmen nur unwesentlich zum Mindestjahreseinkommen beitragen, wählt das elektronische Schieberegister automatisch die nächstteurere Behandlungsmethode, im Zweifel Zahnersatz. Der Onkel Doktor kann sich voll auf "PriDent" verlassen und gewinnt wertvolle Zeit für die zahnärtztlichen Kernaufgaben: Fortbildungssymposien in Honolulu, Ersinnen neuer Steuerstrategien, Auswahl geeigneter Sprechstundenhilfen.

Haben Sie als Unternehmer Probleme mit roten Zahlen? Dann sollten Sie sich für die "2008/14" von Siemens entscheiden, Halle 2. In den ausgeklügelten Programmen auf Basis MSDOS 3.3 sind alle Entscheidungsabläufe abgelegt, die zum Zusammenbruch der Nixdorf AG geführt haben. Kurz, die "2008/14" konzentriert ein enormes Wissen, wie man es nicht macht...

Der "FungTionDong" aus Rotchina ist der billigste Computer der Welt, der bei Vorführungen freilich alle 7 Minuten streikte - möglicherweise das ausschlaggebende Motiv für den Großauftrag des DGB, der gleich 10 000 Stück orderte.

Der "Micky" von IBM spricht die Zielgruppe der Beamten an: Dessen langsamer, aber robuster 4bit-Prozessor benötigt zwar eine Hochlaufzeit bis nach dem Frühstück, überrascht indes durch seine vielen mittels Diskette austauschbaren Module. Zum Beispiel für Postler (Verwaltung mit sich selbst), Lehrer (Verwaltung aller anderen außer sich selbst), Fiskalbeamte (Verwaltung der Finanzverwaltung). Das Modul für Politiker ist besonders interessant; es erstellt Buffetvorschläge für Empfänge und Dementis zu Steuererhöhungen, informiert über Angebote privater Reisen durch die Industrie und liefert Eckdaten für die Diätenerhöhung.

Toshiba präsentiert in Halle 6 den "Supermini", der kleiner ist als ein Taschenrechner, jedoch fünfmal soviel leistet, wie herkömmliche Computer. Seine stündlich abgegebenen Impulse (Fotoblitz oder Nebelhorn) erleichtern die eventuelle Suche nach dem Winzling, sollte er dem Anwender mal im Papierstapel untergehen. Nichtsdestotrotz wird "Supermini" ein Flop: Niemand mag die Tastatur trotz mitgelieferter Lupe bedienen, ganz zu schweigen von der viel zu komplizierten Software. Räsonniert ein Firmensprecher:"Vermarkten wir ihn erst mal in Japan und bringen ihn in zehn Jahren wieder nach Europa".

Gefunden in der FRANKFURTER RUNDSCHAU von H. A. Wuttke

HP-41 MCODE auf dem PC

Advanced Software Development Tools

Benötigte Hardware: HP-41 C/CV/CX, IL-Modul, IBM-Kompatibler Computer, IL-PC Karte oder ein IL-RS232 Interface, Modulsimulator (Rambox oder ähnliches).

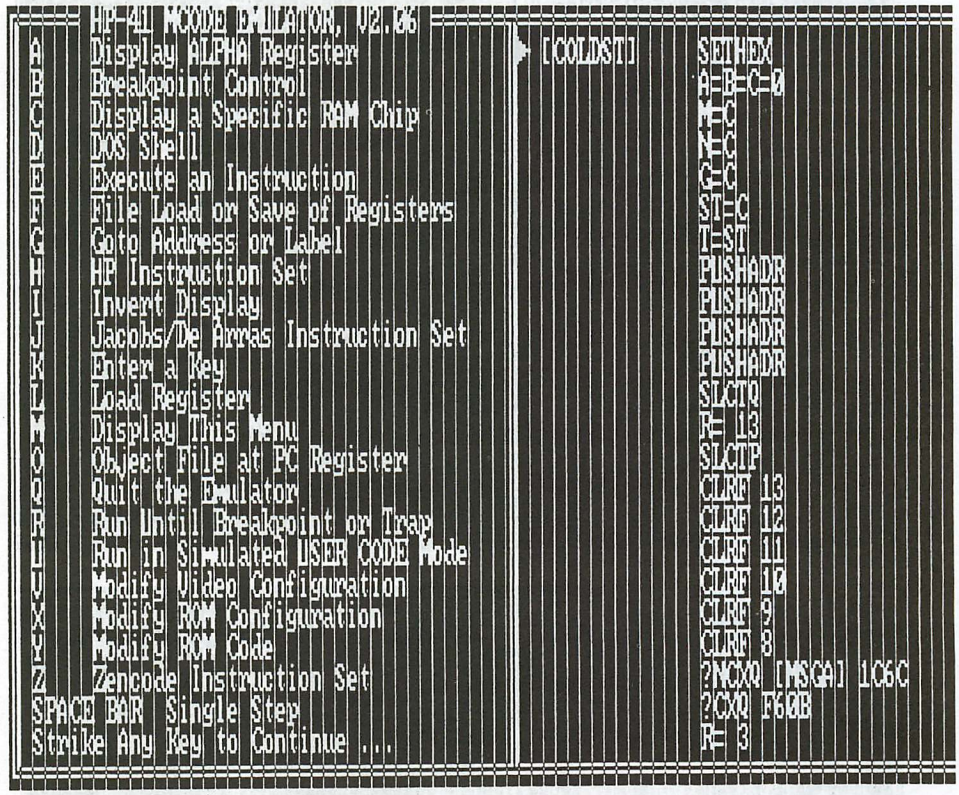
Dieser Artikel soll kein Testbericht sein, dazu fehlt mir momentan noch die Hardware (PC), ich möchte das Programmpaket anhand der mir vorliegenden Unterlagen kurz beschreiben und auf die Funktionen und Besonderheiten eingehen.

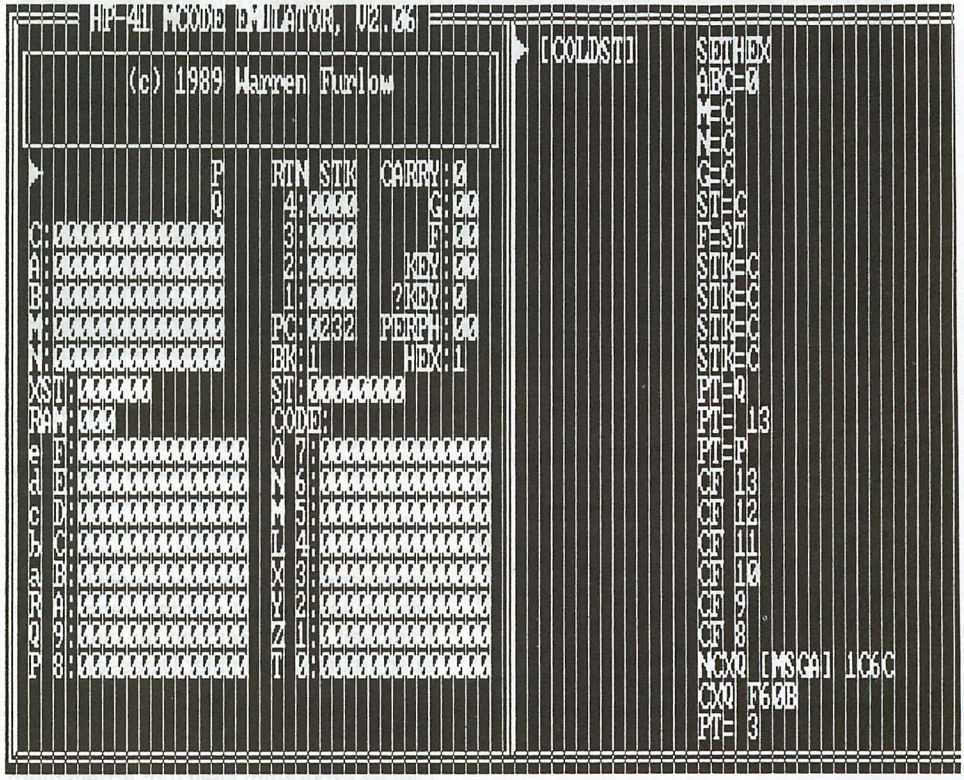
Seit Ende 1989 ist ein professionelles HP-41 Maschinensprachesystem für IBM-Kompatible Rechner zu einem günstigem Preis (in den USA $ 55) erhältlich:

Die Advanced Software Development Tools (ASDT) Release 3.0.

Einige MCODE-Entwickler in den USA benutzen dieses Entwicklungspaket bereits, das auf einer 5 1/4" oder 3 1/2" Diskette geliefert wird. Die mir zur Ansicht zugesandte Vorversion dieses Paketes beinhaltet auch das Handbuch zum Selbstausdrucken, welches aber bei dem im Handel erhältlichen Versionen als richtiges Handbuch mitgeliefert wird.

Nun eine Beschreibung der Einzelprogramme des Paketes:

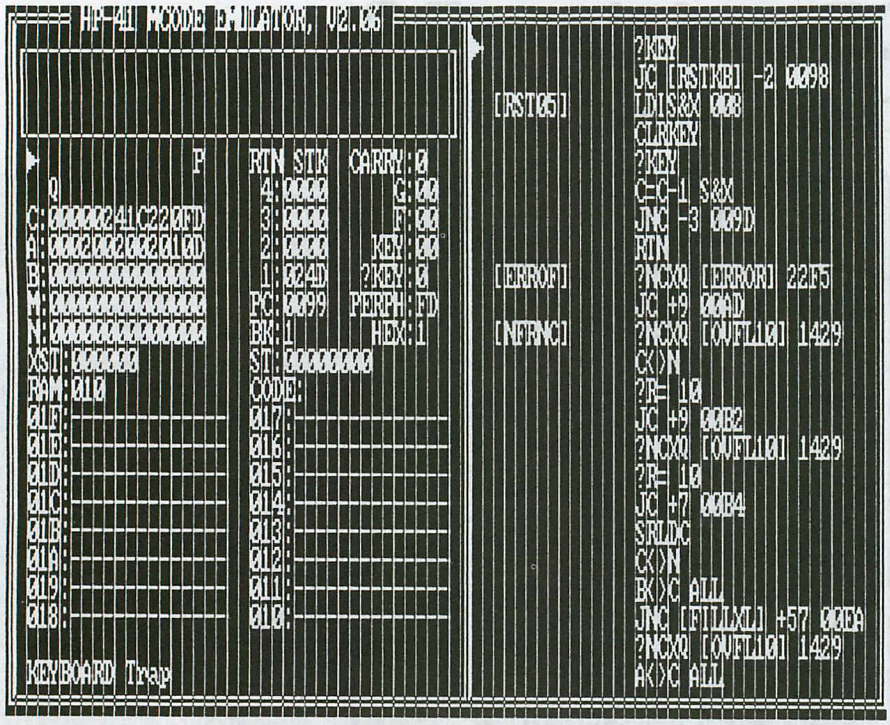


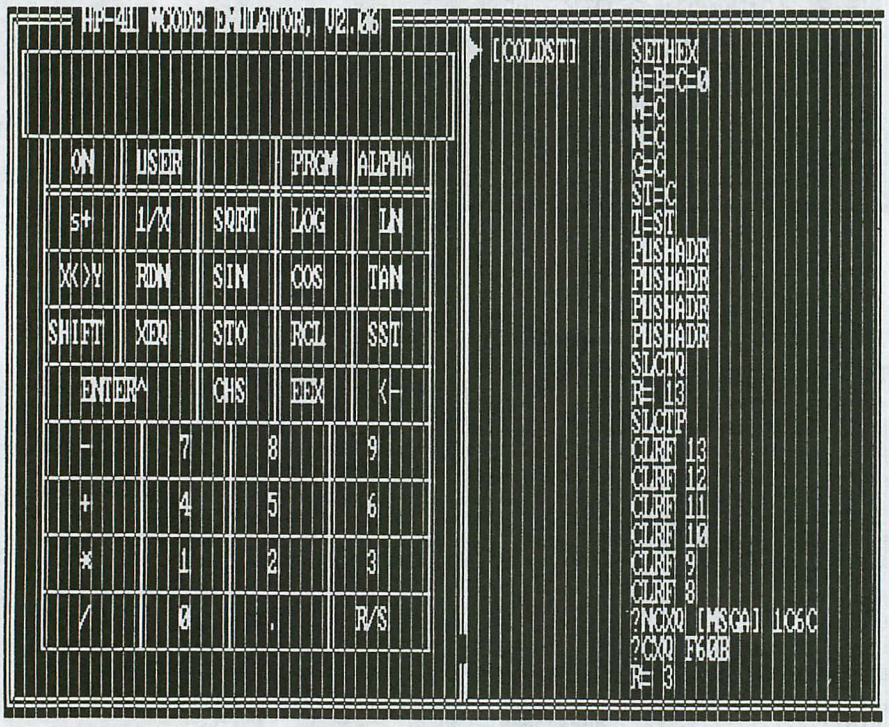


Assembler

Dieser Assembler "versteht" alle drei gebräuchlichen Mnemonics, die bisher verwendet wurden: Jacobs/DeArras, das vom Assembler 3 Eprom oder vom David-Assembler her bekannt ist, ZENCODE der im Zenrom benutzt wird und HP-Mnemonics, die den in den NOMAS Listings verwendeten entsprechen. Durch die Unterstützung dieser drei Mnemonics wird der Umstieg von den HP-41 gestützten Assemblern wesentlich erleichtert. Es braucht also niemand umlernen, falls er von Zenrom oder David-Assembler auf das ASDT aufsteigen möchte.

Eine Tabelle mit den Mainframe-Labels wie sie auch zum David-Assembler erhältlich sind, ist natürlich auch enthalten. Dadurch werden Programme mit Sprüngen in das HP-41 Betriebssystem leichter les- und schreibbar, da bei Sprunganweisungen aüßer vierstelligen Adressen, die sich schwer im Kopf behalten lassen auch alphanumerische Labels eingegeben werden können, die man sich viel leichter merken kann. Einträge in die FAT werden selbstständig vorgenommen, eine Referenztabelle kann ebenfalls ausgegeben werden.





Linker

Verknüpft einzelne MCODE-Routinen zu einem ROM-Image, das in einem HP-41 Modulsimulator (RAMBOX, MLDL, HP-41CY o.ä.) geladen werden kann. Dabei sind 4K nicht die Grenze des Speicherbereiches, bei größerem Platzbedarf der Programme wird ein weiteres Image erstellt. Wie schon beim Assembler erwähnt, ist der Ausdruck einer Referenztabelle möglich.

Disassembler

HP-41 Hexadezimalcodes werden von diesem Disassembler wahlweise in Jacobs/DeArras, ZENCODE oder in HP-Mnemonics übersetzt, die Mainframetabelle wird dabei ebenfalls verwendet. FAT Einträge werden richtig disassembliert, damit ist ein einfacheres Entschlüsseln vom ROM-Images möglich. Ein wichtiger Punkt ist, daß ein mittels des Disassemblers erzeugter Quellfile sofort wieder assembliert werden kann. Diese Fähigkeit ist ein Zeichen für einen sehr ausgereiften Assembler/Disassembler.

Emulator

Dieses Programm stellt das Arbeitspferd dieses Systems dar, mit dem ein beliebiges HP-41 System simuliert werden kann. Dazu lädt man mit Hilfe des Übertragungs-Hilfsprogrammes das HP-41 Betriebssystem vom HP-41 in den PC. Alle Rechnervariationen (C, CV, CX) und Modulkonfigurationen (X-Function, Time, IL usw.) können so auf einfache Weise auf dem PC simuliert werden. Dieser Emulator versetzt den Benutzer auch in die Lage, daß Maschinenprogramme auch im Einzelschritt-Modus ausgeführt und entfehlert werden können. Dabei werden alle CPU-Registerinhalte nach jedem Schritt neu angezeigt. Das 'neue' Display des HP-41 mit allen Kleinbuchstaben wird sowohl unter EGA als auch VGA unterstützt. Eine Logitech-Mouse am PC wird ebensfalls voll unterstützt.

Befehlssatz-Übersetzer

Damit lassen sich Jacobs/DeArras Files in ZENCODE, HP-Mnemonics in Jacobs/DeArras übersetzen, um nur zwei Möglichkeiten zu nennen. Eine Übersetzung aller drei Mnemonics in eine andere ist möglich.

Datenübertragung

Damit lassen sich ROM-Images zwischen HP-41 und PC transferieren, dazu ist unbedingt die Grundgeräte-Ausstattung erforderlich.

Soweit zu den Einzelteilen des Programmpaketes. Nun noch ein paar Worte zur mit gelieferten Dokumentation. Inwieweit das mitgelieferte Handbuch von der mir vorliegenden Diskettenversion abweicht, weiß ich nicht, wenn sie ihr entspricht, dann ist es sehr umfangreich. Das Handbuch umfaßt 50 eng bedruckte DIN-A4-Seiten, wobei die ersten 35 die eigentliche Dokumentation sind, der Rest sind Referenztabellen, Mnemonics usw. Ein Vorteil des späten Erscheinens dieses Paketes ist es, daß alle im Laufe der Zeit hinzugekommenen Systemerweiterungen (Bank Switching, Halfnut Display usw.) beschrieben und berücksichtigt sind. Die Unzulänglichkeiten der HP-41 gestützten MCODE-Entwicklungssysteme werden hier weitgehends behoben. Somit kann abschließend gesagt werden, daß das ASDT trotz seines Erscheinens kurz vor dem "Ablebens" des HP-41 eine sinnvolle und interessante Erweiterung des HP-41 Systens bedeutet. Wäre das Paket drei oder vier Jahre früher auf den Markt gekommen, wer weiß, vielleicht würde der 41er noch unter den Lebenden weilen ?

Damit wäre die Kurzbeschreibung des ASDT beendet, ein Testbericht, eventuell von einem anderen HP-41 Mcoder mit PC soll folgen. Für Fragen zu diesem Paket stehe ich gerne zur Verfügung, sollte jemand das ASDT schon verwenden, bitte ich um Kontaktaufnahme.

Thomas Mareis
Cranachstr. 1
8 München 40

Verkaufe:

HP-48SX	DM 550
RAM-Karte 128 kB	DM 450
HP-28S	DM 300
HP-28C	DM 200
HP-19B	DM 200
HP-18C	DM 150

Alles englische Handbücher!
Tel. 06198/337 25

Suche Original-**Handbuch** (oder gute Kopie) für das HP-41 **Advantage**-ROM.

Tel. 07031/247 38 ab 19.00 Uhr.

Verkaufe:

HP-71 B mit IL-Schnittstelle, Mathe-Modul, Kartenleser komplett DM 750.-

IL-Schnittstellenkarte für PC (HP-82973A) und IL-Link Programm (HP-82477A), zusammen DM 450.-

HP-DeskJet RAM-Erweiterung 256 kB DM 350.-

Tel. 0431/32 80 32

Thermodrucker HP 82143 A für DM 120.-
Christian Grotkamp, Heidbergweg 6, 4300 Essen 15, Tel. 0201/48 34 29 oder 0208/43 05 37.

Verkaufe:

HP-28C/S	DM 350
HP ThinkJet IL	DM 400

Volker Lang, Weilerweg 37, 7103 Schwaigern, Tel. 07138/5349 ab 18.00 Uhr.

Verkaufe:

HP-41CV	DM 300
IR-Modul	DM 100
Magnetkartenleser HP 82104 A	DM 300
Mathe I Modul	DM 50
X-Funktion Modul	DM 80
Thermodrucker 82240 A	DM 180

Tel. 05407/2923

Suche: HP-41 CV oder CX, ggf. auch mit Zubehör. Preis VS. Angebote an Rösner, Schöne Aussicht 9, 3501 Schauenburg, Tel. 05601/4368 ab 17.00 Uhr oder BTX 0560143680001.

Verkaufe:

HP 75 C + 8 k Erweit.	DM 300
HP 28C	DM 130
HP 82240 A Infrarot Taschendrucker	DM 110
HP 41 CV + X-Funktion + 2 Memory Module	DM 300
HP 82160 A IL-Modul	DM 120
HP 82161 A Kassettenlaufwerk + Leerkassetten	DM 400
HP 82162 A Thermodrucker	DM 250
HP 82905 B A4 Printer	DM 300
HP 41-71 Translator	DM 120

Alle Geräte mit Handbüchern.

Günter Vollmer, 07472/8245 ab 18.00 Uhr.

HP Serie 80: Wer hat noch Geräte und Zubehör (Rom's, Module etc.), die er nicht mehr braucht?Bitte nachsehen und bei W.Klos, Tel. 06427/8032 melden.

Hiermit biete ich zum **Verkauf** an:

- Marys II (CMT MC-II) mit 256 kB RAM
- HP-IL interface und parallele Schnittstelle für MC-II
- 1 MB RAM-Disk
- POD-TRANS (Übertragungs und Übersetzungsmodul für HP-41 Programme)
- HP-41 Emulation
- Die fast vollständige Software, die es für den HP-41 gibt, befindet sich auf der RAM-Disk!

Alle Teile sind in einem absolut neuwertigen Zustand
Komplettpreis für das MC-II Paket DM 3900.-
1 HP 28SD mit diverser Fachliteratur komplett in deutsch DM400.-
1 CASIO fx-8000G, grafikfähiger Taschencomputer mit 2,5 kB DM170.-
Viel Fachliteratur für den HP-41 auf Anfrage.
K.-U. Peikert, Tel. 0421/47 00 26.

Verkaufe: **HP-75 Forth**
Compiler bzw. Interpreter, Editor, Assembler, Debugger und Disassembler incl. Source; lauffähig mit und ohne RAM-Modul, 1 Kassete + Handbuch + Lehrbuch DM 170.-
Dieser Preis ist Ihr Endpreis incl. Poro und Verpackung.
Peter Habicht, Tel. 06008/7235 ab 19.00 Uhr.

HP-71B Systemzubehör:

*	Kartenleser für HP-71B mit Magnetkarten	DM 190
*	Kassettenlaufwerk mit Netzteil	DM 390
*	PAC Screen, einfach	DM 90
*	12" Monitor bernstein, passend für PAC Screen	DM 130
*	HP41 Tranlator ROM	DM 90
*	AC-Circuit Anal. Pac	DM 90

Alle IL-Geräte mit original Handbüchern und je 1 IL-Kabel.
Walter Becker, Buschkämpen 41, 2850 Bremerhaven, Tel. 0471/565 53 abends.

Suche dringend:
- HP 71B
- Mathemodul für den HP 71B
- HPIL Thermodrucker

Angebote an:
Tobie Niggli, Martin Disteliestraße 97, CH-4600 Olten, Tel. Schweiz 062/26 32 03 oder 062/26 33 13.

Verkaufe HP 82240 A Thermodrucker mit Netzteil DM 200
Michael Pulm, Tel. 02461/4918.

Suche HP 3468 Multimeter mit HP-IL Schnittstelle und HP 7475 Plotter mit RS-232 Schnittstelle.
Heinz Schmitt, Tel. 02845/6204 nach 18 Uhr.

Verkaufe:

HP 71 mit IL-Modul	DM 500
96 k RAM	DM 400
Mathe-ROM	DM 120
Forth/Assembler-ROM	DM 90
HP41-Translator-ROM	DM 100
Schaltkreisanal.-ROM	DM 50
HP 9114 Diskettenlaufwerk mit W&W-Netzteil	DM 550
HP-ThinkJet	DM 490
HP 82166 IL-Converter	DM 250
Advanced PACSCREEN Videointerface mit Monitor und Maus	DM 370

dazu **reichlich Software, Dokumentation und Zubehör**.
Komplettpreis DM 2400

32 k RAM	DM 200
HP-Kassettenlaufwerk	DM 350

Joachim Kühn, K.-Kreiten-Straße 5, 5300 Bonn 1, Tel. 0228/65 00 89 privat, 0228/382 66 51 dienstl.

HP 150-2-System, bestehend aus: HP 150-2, 512 kByte, Touch-Screen-Option, Doppel-Disc-Drive HP 9221, Keyboard, Printer IB-ThinkJet, Software, komplette Dokum., DM 800.- VB.
HP 125 mit kleinen Fehlern, DM 200.- VB
Taschenrechner **HP 32S** (Softkeys) ohne Buch, DM 50.-
SIMM (Single Inline Memory Module) 256 kByte, 80 ns, 9 Bit
Gysbert Hagemann, Alter Weg 1, D-6653 Blieskastel 2, Tel 06842/2805, geschäftszeitlich: 06842/3041/3042.

Suche für das HP-IL-System
Diskettenlaufwerk HP 9114
Grabau GR7 Video-Controller
512 kByte CMT-Rambox
Wer leiht mir gegen Honorar **JPC-HandBuch** ?

Dr. H. Waldmann, Schwendlerweg 19, 7959 Kirchberg, Tel. 07354/8729.

Zu Verkaufen:

*** Serie 70 ***

-	HP-71B mit Kartenleser 82400, neuw.	DM 750
-	Kartenleser 82400A, neu	DM 120
-	HP-41 Translator Pac 82490A, neu	DM 100
-	Kurvenanpassung 82484AD, neu	DM 150
-	30 Blank Cards, neu	DM 60
-	HP-75D, 8 k RAM-Modul, 64 k Expansionpod., opt. Lesestift, gebr.	DM 800

*** Serie 40 ***

-	HP-41C mit Quad-Memory-, Time-Modul, gebr.	DM 175
-	Time Modul 82182A, neu	DM 50
-	Drucker 82143A, neuw., orig. verp.	DM 250
-	Drucker 82143A, gebr.	DM 175
-	Accu Pac für HP-41 82120A, gebr.	DM 30
-	Mathe Modul 00041-15011, neu	DM 30
-	Games Pac 00041-15022, neu	DM 40

*** HP-IL Geräte ***

-	Diskettenlaufwerk 9114B, neuw.	DM 790
-	Cassettenlaufwerk 82161A, neu	DM 550
-	Cassettenlaufwerk 82161A, neuw., orig. verp.	DM 450
-	ThinkJet 2225B, gebr., orig. verp.	DM 480

Alle Geräte mit Handbüchern, original Zubehör.

Thomas Mareis, Cranachstraße 1, 8000 München 40, Tel. 089/129 56 65.

Impressum

Titel:
PRISMA

Herausgeber:
CCD - Computerclub Deutschland e.V.

Postfach 11 04 11
Schwalbacher Straße 50
6000 Frankfurt 1

Verantwortlicher Redakteur:
Alf-Norman Tietze (ant)

Redaktion:
Michael Krocker (mik)
Martin Meyer (mm)
Dieter Wolf (dw)

Herstellung:
CCD e.V.

Manuskripte:
Manuskripte werden gerne von der Redaktion angenommen. Honorare werden in der Regel nicht gezahlt. Die Zustimmung des Verfassers zum Abdruck wird vorausgesetzt. Für alle Veröffentlichungen wird weder durch den Verein noch durch seine Mitglieder eine irgendwie geartete Garantie übernommen.

Druck und Weiterverarbeitung:
Reha Werkstatt Rödelheim
Biedenkopfer Weg 40 a, 6000 Frankfurt

Anzeigenpreise:
Es gilt unsere Anzeigenpreisliste 3 vom Juni 1987

Erscheinungsweise:
PRISMA erscheint jeden 2. Monat

Auflage:
2500

Bezug:
PRISMA wird allen Mitgliedern des CCD ohne Anforderung übersandt. Ein Anspruch auf eine Mindestzahl von Ausgaben besteht nicht. Der Bezugspreis ist im Mitgliedsbeitrag enthalten.

Postvertriebsstück
Gebühr bezahlt

D 2856 F

CCD - Computerclub Deutschland e.V
Postfach 11 04 11
D-6000 Frankfurt am Main 1

CCD
ISSN 0176-8735
PRISMA
Januar/Februar 1991