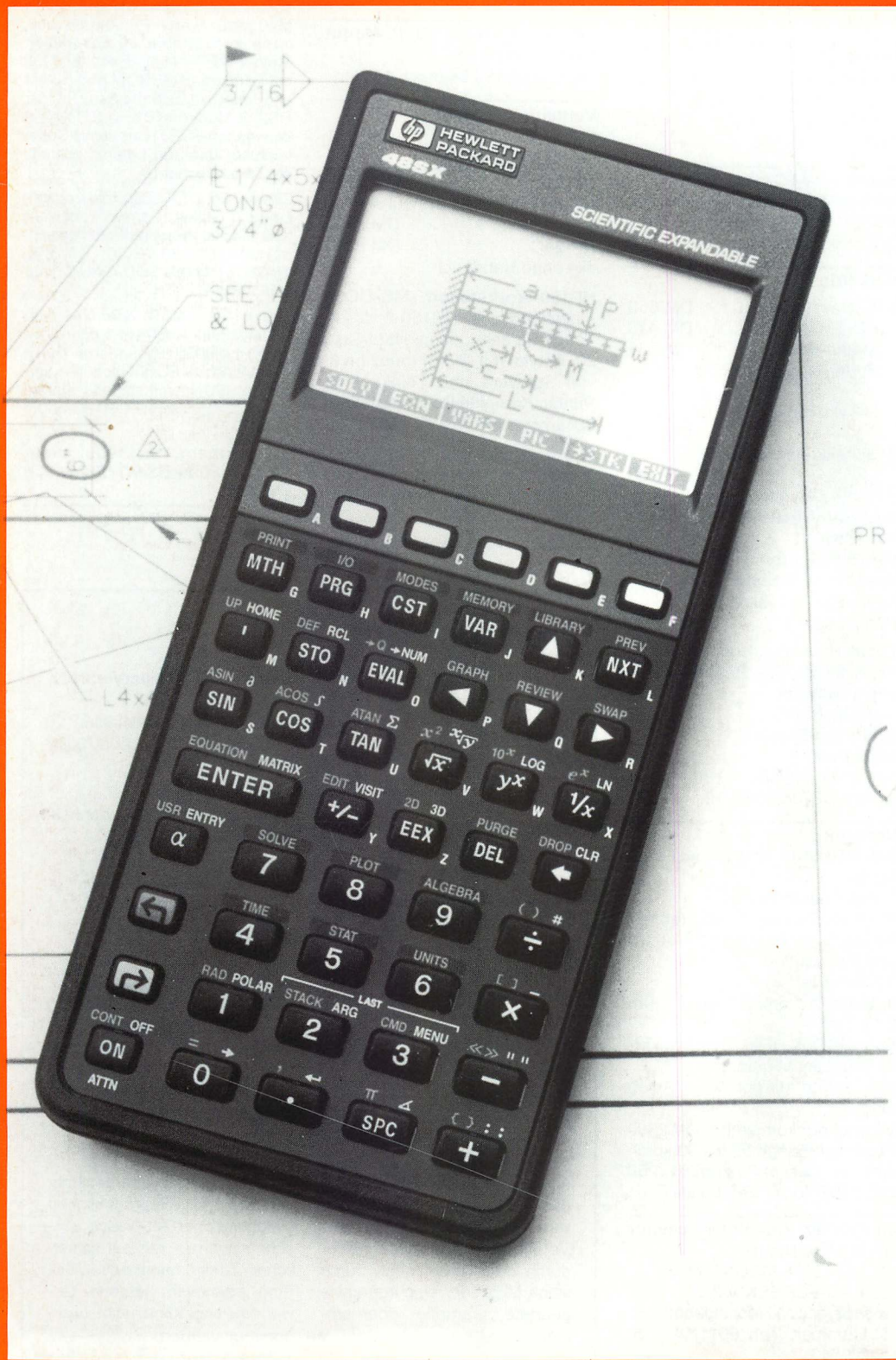
PRISMA

Computerclub Deutschland e.V. · Postfach 11 04 11 · Schwalbacher Straße 50 · D-6000 Frankfurt am Main 1

Juli /August 1990 Nr. 4 **D 2856 F**



Mit dem 48SX setzt Hewlett-Packard einen weiteren Meilenstein in der Taschencomputer-Geschichte.

Seine grafischen Fähigkeiten bilden den Grundstein für die aussagekräftige Mensch-Maschine Kommunikation.

Der CCD wird diese Fähigkeiten verstärkt unterstützen - ein Beitrag mehr zur Förderung von Technik und Wissenschaft.

Clubbörse

HP-71B System:

* Kartenleser für HP-71B mit Magnetkarten DM 220
* Kassettenlaufwerk mit Netzteil DM 490
* PAC-Screen, einfach DM 120
* Advanced PAC-Screen mit Netzteil DM 240
* 12" Monitor bernstein, passend für PAC-Screen DM 150
* 2 Paar IL-Kabel 5m je DM 35
* HP41 Translator ROM DM 120
* AC-Cicuit Anal. Pac DM 120

Alle IL-Geräte mit original Handbüchern und je 1 IL-Kabel.

Walter Becker, Buschkämpen 41, 2850 Bremerhaven, Tel. 0471/56 553 (abends).

Verkaufe:

HP 28 C DM 130
HP 71 B mit IL-Modul DM 300
Diskettenlaufwerk
HP 9114 A DM 500
komplett nur DM 850

R. Mulch, Tel. 06154/52 689 abends.

Verkaufe HP-41CX

* HP-41CX DM 305
* Kartenleser 82104A DM 155
* CCD-Modul 82500B DM 110
* Erw. Modul 82106A DM 20
* IL-Modul 82160A DM 120
* Netzteil 82066 DM 20
* IL-Kabel DM 5
* 15 Progamme HP-USER's mit Magnetkarten DM 120
* Wickes: Synthetische Programmierung DM 15

- alles inkl. kompletter Dokumentation.

Verkaufe HP-71B

* HP-71B + IL-Modul DM 600
* Kartenleser 82400A DM 175
* Texteditor 7240 DM 125
* 71/41 Translator 7269 DM 120
* IL-Modul 82401A DM 120
* Tatzl: Prakt. Anwendungen mit dem HP-71B DM 15
* J. Gruss: Textverarbeitung HP-71 + Disk 3 1/2" DM 50

- alles inkl. kompletter Dokumentation.

* Prisma 83-90 fast komplett DM 110 bzw. DM 3,50/Stck.

Peter Klaiber, 2100 HH 90, Dahlengrund 10, Tel. 040/760 73 66 (abends).

Verkaufe

Microsoft **WINDOWS 3.00**
Deutsch, 5 1/4 Zoll, 1.2 MB, Handbuch, original versiegelt DM 230

HP-75 Zubehör
RAMerweiterung um 8k DM 190

HP-75
Programmsammlung **FIGForth**
1 Kassette eigenes Betriebssystem DM 180

Alle Preise sind Ihre Endpreise incl. Porto und Verpackung
Peter Habicht, Tel. 06008/7235 ab 19.00 Uhr

Welche **HP-28/48SX** Besitzer im Raum **Freiburg/Brsg.** sind an einem Erfahrungs- und Gedankenaustausch interessiert?
Interessenten melden sich bei: Gregory Kuhnmunch; Grünwälderstraße 2, 7800 Freiburg/Brsg., Tel. 0761/38 34 75.

Verkaufe:
IL-Schnittstellenkarte für PC (HP-82973A) und IL-Link Programm (HP-82477A) zusammen DM 500
HP-DeskJet RAM-Erweiterung 256 KB DM 400
Tel. 0431/32 80 32.

HP-41CX
Kartenleser, Thermodrucker, 2 Ladegeräte, Time-Modul, alle Handbücher VB DM 750
Tel. 02603/13 565.

Suche **HP-41CX** und IL-Modul
Herbert Rothe, Stöckstraße 12, 7570 Baden-Baden, Tel. 07221/53 350.

Verkaufe:

HP-71 mit IL-Modul DM 550
96k RAM DM 420
Mathe-ROM DM 130
Forth/Assembler-ROM DM 100
HP41-Translator-ROM DM 120
Schaltkreisanalyse-ROM DM 80
HP9114 Diskettenlaufwerk mit W&W-Netzteil DM 550
HP-ThinkJet DM 500
HP82166 IL-Converter DM 250
dazu reichlich Software, Dokumentation und Zubehör.
Komplettpreis DM 2300

Zusätzlich verkaufe ich HP-71 mit IL-Modul, 32k-RAM und Kassettenlaufwerk.
Joachim Kühn,
K.-Kreiten-Straße 5,
5300 Bonn 1,
Tel. privat 0228/65 00 89,
dienstl. 0228/382 64 19.

Verkaufe:
HP-71B mit IL-, Mathe- und Forth-Modul.
Tastenfeldschablonen.
ThinkJet-Drucker (IL) mit Netzteil.
Kassettenlaufwerk mit Netzteil und 5 Kassetten.
IL Converter.
IL Kabel: 1 * 5m, 2 * 2m, 2 * 1m, 2 * 0,5m.
Bücher: Das HP-IL-System, HP-71 Basic leicht gemacht.
Alle Geräte mit deutscher Anleitung.
Verkauf nur komplett! DM 2300
Prismajahrgänge 82 bis 90 komplett und Best of Prisma DM 260
Tel. 0202/70 26 18 (abends).

80-Zeichen Videointerfaces für HP-IL zu verkaufen:

1. HP 92198 MOUNTAIN
2. PAC-SCREEN V 2.0

Abgabe gegen Höchstgebot.
M. Hammer, Tel. 0611/54 138, CCD 2743

Verkaufe:
HP-28SD DM 320
HP15C (auch Matrizen!) DM 110
HP-41-Tastenfeldschablonen je DM 2
Rechner und Handbuch sind wie neu.
J. F. Graf, Perelsstraße 21, 2050 Hamburg 80, Tel. 040/730 31 57

Suche CCD-Modul von W&W (Version B) für HP-41CX.
Stefan Schneider, Gabelsbornstraße 30, 6200 Wiesbaden, Tel. 0611/84 09 83.

Suche für HP-41C **IL-Modul** 82160A, Preis VS.
Tel. 09281/187 54 (nach 18 Uhr).

Verkaufe CCD-Modul
für HP-41 DM 150
Tel. 0211/62 43 06.

HP-41C DM 100
HP-18C Business Consultant DM 100
HP-28C DM 200
alle ohne Handbuch.

HP150-Workstation (MS-DOS) bestehend aus: HP150-2, 512k, Touch-Screen-Option, Hard-Disc 20 MByte mit 3.5" Floppydrive, viel Software und ThinkJet-IB-Printer. Komplte Dokumentation.
Zusammen VB DM 1700

Gysbert Hagemann, Alter Weg 1, D-6653 Blieskastel 2, Tel 06842/2805, geschäftszeitlich: 06842/3041/3042.

FORTH,

der Echtzeit90Gewinner.

Im Programmierwettbewerb setzte das Rostocker Team comFORTH ein und gewann mit ihrem 2.5 MHz Z80 Robotron. Im HP48 läuft ein Forth-Dialekt, einen Stack und RPN können CCD'ler im Schlaf bedienen.

Warum dann nicht 'mal FORTH komfortabel nutzen und erlernen, und zwar inkl. Meta- und Target-Compiler, Assembler, Disassembler, Tracer, Floating Point, Editor und (!) Hyper-Help ... , und alles 4th-gemäß natürlich auch im Quelltext . Mit F-PC (c't 90-11 p.226f) schreibe ich Texte, steuere meine Meßgerte, werte Ergebnisse aus, lerne komfortabel 86-Assembler. Mein FORTH-Mini- Rechner läuft mit Forth im E(e)PROM und SRAM und unterhält sich mit dem PC per RS232. Moderne fP's, wie RTX2000 verwenden FORTH als native Code, kürzeste Interrupt Latenz, schnellste Task-Wechsel.

Zimmer Forth F-PC Spezial Version, Demo (1 Disk, Format ?, jeweils 'voll') DM 20, KomplettPaket gezippt 5*1M44 DM 80 (CCD: DM 60), 15*360k DM 90 (CCD: DM 70), nur Vorkasse.

680xx (Atari, MAC) oder C64 oder CP/M ? Mini-Rechner ? oder C (--uNIX) mit FORTH generieren ? auch da kann ich zu FORTHschritten verhelfen. SUN workstations starten zuerst Diagnosen in FORTH, warum wohl?

Klingelnberg, Strassburgerstr. 12, D-5110 Alsdorf, T. 0 2404 - 6 16 48.

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.

Inhalt

48'er - stark im Kommen

Die Einsendungen von Programmbeiträgen und anderen Beiträgen zum HP-48SX nehmen in der PRISMA-Redaktion stetig zu - ein untrügliches Indiz für gestiegenes Interesse an diesem neuen und zugleich neuartigen Taschencomputer. Alles deutet darauf hin, daß sich der HP-48SX auch ohne "operative Hektik" zum neuen Schwerpunkt im PRISMA entwickelt. Ein gutes Dutzend Seiten konnten in diesem Heft mit 48'er Themen belegt werden. Das spricht für sich.

Und wir haben noch weitere Artikel in der redaktionellen Aufbereitung - z.B. ein interessantes Datenbank-Grundprogramm von Martin Breidenbach aus Bad Vilbel für die persönliche "Überarbeitung", das eigentlich gar kein Programm ist, sondern eher eine raffinierte Tastaturbelegung. Mehr darüber im nächsten Heft.

Aber auch für die anderen Rechnersysteme ist nach wie vor Platz im PRISMA, sofern zur Veröffentlichung bestimmte Artikel bei uns eintreffen. Für zwei unterschiedliche Rechner zugleich befindet sich in dieser Ausgabe ein im übertragenen Sinne "duftendes" Thema. Es darf geraten - oder besser: gesucht - werden.

Verschenktes Geld?!

Den Schritt vom Schreiben zum Kassieren kann der CCD e.V. nur indirekt verkürzen, da uns für den direkten Weg der Bezahlung von Autoren nach wie vor keine Mittel zur Verfügung stehen, woran sich auch in absehbarer Zukunft nichts ändern wird. Indirekt aber gibt es einen neuen, alten und häufig ungenutzten Weg, doch noch einen kleinen Betrag für die geleistete Autorenarbeit "abzuschöpfen".

Das Zauberwort heißt "Verwertungsgesellschaft". Über eine solche - in unserem Falle über die Verwertungsgesellschaft Wort - kann man als "wahrgenommener" (d.h. registrierter) Autor an den auf unterschiedlichste Weise eingenommenen Schutzgebühren für beliebige Zweitnutzungen partizipieren. Um allerdings keine falschen Hoffnungen zu wecken sei klargestellt, daß diese Regelung nur bei nennenswerten Veröffentlichungen bzw. Artikeln einen Nutzen bringt. Ein kleiner Fünfzeiler "Hilfe - mein Drucker druckt keine Graphik" oder ähnliches ist mit Sicherheit nicht für eine Registrierung geeignet. Zumindest für unsere Vielschreiber dürfte diese Regelung interessant sein. Na, neugierig geworden? Der Artikel "Die Goldesel" von Goetz Buchholz in diesem Heft liefert die Antworten.

Alf-Norman Tietze
(Chefredakteur)

Die Goldesel

Verwertungsgesellschaften von der GEMA bis zur VG Wort

Von Goetz Buchholz

Mit freundlicher Genehmigung des Autors drucken wir hier einen Artikel aus "Publizistik und Kunst", einer Mitgliederzeitschrift der IG Medien, ab. Der Artikel dürfte besonders unsere Autoren interessieren. Die vorliegende Fassung ist um die Teile gekürzt, die für PRISMA-Autoren nicht relevant sind, wie z.B. die Verwertungsgesellschaften für Musik und Film. (dw)

Neun Verwertungsgesellschaften verhelfen in der Bundesrepublik Schriftstellern und Fotografinnen, Kameraleuten und Malern, Verlegern, Musikerinnen, Journalisten und Filmproduzenten einmal im Jahr zu einer unverhofften Einnahme: Der Tantieme für die Zweitnutzungen ihrer urheberrechtlich genutzten "Werke". Obwohl der jährliche Scheck ohne jedes Kostenrisiko und mit relativ wenig Bürokratie zu haben ist, verzichten immer noch Tausende von Autoren auf dies Geld.

Wenn es sie nicht gäbe, müßte man sie erfinden: Die Verwertungsgesellschaften. Man schickt ihnen ein paar Formulare, und bekommt im Gegenzug einen Scheck. Für nichts und wieder nichts, so stellt es sich vielen dar.

Sie gehören zu den bestgehaßten Institutionen. Für alles und jedes verlangen sie Geld, allen voran die GEMA, für jede Platte, die im Radio gespielt, jedes Lied, das auf dem Schulfest gesungen, jede Seite die aus einem Buch kopiert wird.

Sie sind eine Folge der technischen Entwicklung. Dank der Fotokopierer existieren wissenschaftliche Bücher heute oft in mehr kopierten als Original-Exemplaren, dank perfekter Tonbandtechnik wird kaum ein Walkman mit Original-Aufnahmen betrieben, der Wiederholung sämtlicher Folgen der Schwarzwaldklinik im Wohnzimmer sind dank Videorecorder keine Grenzen mehr gesetzt. Und die Autoren, Musikerinnen und Kameraleute hätten nichts davon. Wenn es die Verwertungsgesellschaften nicht gäbe.

PFENNIGHONORARE UND MILIONENBETRÄGE

Da hat eine Journalistin für die Zeitschrift "natur" einen Artikel zum Gülleproblem im Südoldenburgischen geschrieben. Sie hat vom Verlag ihr Honorar bekommen, der Artikel wird gedruckt - und längst nicht nur von den Käufern der Zeitschrift gelesen: Die Landesregierung druckt den Beitrag im Pressespiegel nach; die betroffene Gemeinde hängt ihn im Dorf-Schaukasten aus; ein Lehrer kopiert ihn dreißigmal, um ihn mit seiner Klasse zu besprechen; die Zeitschrift wird in Bibliotheken ausgeliehen und in Lesezirkeln vermietet. Lauter kostenpflichtige Nutzungen, die im Kaufpreis der Zeitung (und dem Abdruckhonorar vom Verlag) keinesfalls enthalten sind.

Da aber die Journalistin ebenso wie der Verlag hoffnungslos überfordert wäre, sollte sie solche Nutzungen überhaupt nur feststellen oder gar Honorare dafür eintreiben, beauftragt sie damit eine Verwertungsgesellschft. In diesem Fall die VG Wort, die im letzten Jahr über 60 Millionen Mark an solchen Pfennighonoraren eingesammelt und an rund 50 000 Autoren - immerhin rund 500 Mark pro Kopf - sowie 2500 Verlage ausgezahlt hat.

Eingetrieben wird das Geld auf unterschiedlichste Weisen:

- Wer einen Pressespiegel herausgibt, muß dafür einen kostenpflichtigen Vertrag mit der VG Wort abschließen.
- Wer Fotokopiergeräte gewerblich betreibt, muß dafür eine Abgabe zahlen; für ihre Schulen zahlen die Länder eine pauschale Fotokopiervergütung.
- Wer einen Lesezirkel betreibt, zahlt eine Abgabe; auch öffentliche Bibliotheken führen ihren "Bibliotheksgroschen" ab.

BÜROKRATIE ZAHLT SICH AUS

Für die "Urheber der Werke", wie sie im Gesetz nun einmal heißen, ist nichts leichter, als sich ihren Anteil an diesen Geldern zu sichern. Sie brauchen nur einen "Wahrnehmungsvertrag" mit der jeweils zuständigen Verwertungsgesellschaft zu unterschreiben (Formular wird auf Anforderung zugeschickt), in dem sie die entsprechenden Rechte abtreten, und der Gesellschaft regelmäßig die von ihnen produzierten "Werke" zu melden. Diese Meldungen sind die Grundlage für die jährliche Geldverteilung - ein bißchen Bürokratie lohnt sich also: Wer wenig meldet, bekommt wenig.

Die Modalitäten sind bei den einzelnen Verwertungsgesellschaften verschieden und können hier im Detail nicht aufgeführt werden.

Einheitlich gilt aber für alle Verwertungsgesellschaften: Sie müssen einen Wahrnehmungsvertrag mit jedem abschließen, der das verlangt (und müssen jedem der - gegen "angemessenes Entgelt" - die Nutzung der von ihnen betreuten Werke erlauben).

WELCHE RECHTE HABEN DIE URHEBER?

Das "Gesetz über die Wahrnehmung von Urheberrechten und verwandten Schutzrechten" setzt die für die Gründung einer Verwertungsgesellschaft (VG) mehrere Bedingungen. Die wichtigsten aus Sicht der Urheber sind:

- Niemand darf abgewiesen werden - die Verwertungsgesellschaft muß einen Wahrnehmungsvertrag mit jedem abschließen, der das verlangt (und Urheberrechte im jeweiligen Bereich besitzt). Und zwar "zu angemessenen Bedingungen" - das heißt in der Regel kostenlos.
- Die Geldverteilung muß kontrollierbar sein - die Verwertungsgesellschaft hat einen festen Verteilungsplan aufzustellen, der "ein willkürliches Vorgehen ausschließt".
- Förderung der Kultur - der Verteilungsplan "soll dem Grundsatz entsprechen, daß kulturell bedeutende Werke und Leistungen zu fördern sind".
- Hilfe für Pechvögel - die Verwertungsgesellschaften "sollen" Vorsorge- und Unterstützungseinrichtungen schaffen. In der Regel sind das "Sozialfonds", die "unverschuldet in Not geratene Urheber" unterstützen. Die VG Wort zahlt darüber hinaus freien Schriftstellern und Journalisten Zuschüsse zur Lebens- und Krankengeldtageversicherung.
- Aufsichtbehörde ist das Patentamt.
- Internationale Kooperation - die Verwertungsgesellschaften nehmen nur Deutsche "im Sinne des Grundgesetzes" und Leute auf, die "im Geltungsbereich dieses Gesetzes" wohnen. Vergütungen für Nutzungen im Ausland werden von den dortigen Verwertungsgesellschaften eingezogen und miteinander verrechnet.

Für alle "Worturheber" - vom Journalisten bis zur Kinderbuchautorin - ist die VG Wort zuständig.

WER IST ZUSTÄNDIG?

VG Wort

Verwertungsgesellschaft Wort, Goethestraße 49, 8000 München 2.

Wahrnehmungsberechtigt: Autoren und Übersetzer schöngeistiger, dramatischer, wissenschaftlicher, Sach- und Fachliteratur sowie Journalisten.

,..UND NOCH EIN LEXFILE:

ENDUPLEX

von Micheal Fiedler

Bei diesem Lexfile handelt es sich um eine Ergänzung des HP-71B Betriebssystems.

Bekanntlich gibt es den STARTUP-Befehl, und hier in diesem Lexfile befindet sich das genaue Gegenstück dazu.

Der mit STARTUP gesetzte String wird bei jedem Einschalten des Rechners automatisch ausgeführt. Analog verhält es sich hier mit dem Befehl ENDUP. Bei jedem Ausschalten des Rechners wird ein beliebiger String ausgeführt. Zusätzlich besteht nun die Möglichkeit, sich die einmal gesetzten Strings wieder anzeigen zu lassen.

Der französische Kollege Jean-Jacques Moreau hat diesen Lexfile geschrieben und er wurde in JPC 31, dem Journal des PPC Paris, in seiner jetzigen Fassung vorgestellt.

Die Filegröße ist 304 Bytes (286 laut CAT), es sind 4 Schlüsselwörter vorhanden:

```
ENDUPLEX ID=E1   SIZE=304 Bytes
   Polls!
 10  ENDUP$     XFN    225016 (-)
 11  ENDUP      Stmt   225017
 12  STARTUP$   XFN    225018 (-)
 13  EXECUTE    Stmt   225019
*
```

Hier die Erläuterungen zu den einzelnen Schlüsselwörtern:

ENDUP$

Damit kann man den einmal gesetzten Befehlsstring wieder abfragen. Ist noch keiner gesetzt, so wird ein Leerstring zurückgegeben.

Beispiel:

```
IF NOT LEN(ENDUP$) THEN DISP
"Kein ENDUP-String gesetzt!"
```

ENDUP *string*

In diesem String kann ein, oder auch mehrere Befehle stehen (durch @ getrennt), die zur Ausführung (beim Ausschalten des Rechners) wie eine Programmzeile abgearbeitet werden.

Beispiele:

```
ENDUP "DELAY 1,0
      @ DISP 'TSCHÜSS, BIS BALD!'"
```

oder, falls Sie mit LOCK ein Paßwort vereinbart haben:

```
ENDUP
  "DISP 'PASSWORT GEMERKT? J/N'
   @ IF KEYWAIT$ # 'J' THEN LOCK'"
```

oder, falls Sie ein Lexfile zur Uhranzeige benützen, wie etwa CLOCKLEX (das auch bei ausgeschaltetem Rechner unnötig Strom verbraucht):

```
ENDUP "CLOCK OFF"
```

Eine hübsche Schlußmelodie wäre auch denkbar:

```
ENDUP "BEEP2000,.2@BEEP1500,.1
      @BEEP1500,.1@BEEP1550,.2
      @BEEP1500,.4 @BEEP1900,.2
      @BEEP2000,.2"
```

STARTUP$

Dies entspricht genau der Funktion ENDUP$, hier wird aber der mit STARTUP (aus dem Betriebssystem) gesetzte String zurückgegeben.

EXECUTE *string*

Dies ist offensichtlich ein Nebenprodukt der bisherigen Schlüsselwörter. Mit diesem Befehl wird ein beliebiger Befehlsstring direkt abgearbeitet. Im Prinzip handelt es sich hierbei zunächst um nichts anderes, als wenn eine Befehlszeile im Direktmodus, oder eine Programmzeile im Programmlauf abgearbeitet wird.

Das Besondere liegt hier darin, daß mit EXECUTE auch nicht programmierbare Funktionen programmiert werden können!

Beispiel:
Sie habe im Port Nr. 4 ein RAM-Modul und möchten den Port(4) programmgesteuert unabhängig machen:

```
FREEPORT BASIC 166 Bytes, HP-71B

10 EXECUTE 'FREE PORT(4)
                  @ RUN FREEPORT,20'
20 DISP 'PORT(4) OK. MEM=';
                       STR$(MEM(4))
30 END
```

Nach der Ausführung des FREE-Befehls wird zwar der Programmablauf unterbrochen, durch den nachfolgenden RUN-Befehl mit Zeilennummer, kann das Programm jedoch beliebig wieder aufgenommen werden. Es muß lediglich darauf geachtet werden, daß der richtige Programmname (hier FREEPORT) eingesetzt wird.

So wie die meisten anderen Lexfiles der französischen PPC-Autoren, so sind auch die Schlüsselwörter von ENDUPLEX im JPC-Rom enthalten.

Der Lexfile ENDUPLEX ist wie immer entweder durch Abtippen des Listings (mit dem Programm MAKEFILE), oder bei unserer Programmbibliothek (Henry Schimmer, siehe Serviceleistungen), oder über unsere Mailbox GEO1 zu bekommen.

ENDUP gut, alles gut!

Hexdump-Listing ENDUPLEX

(zum direkten Übernehmen benötigen Sie MAKEFILE aus der HP-71B Programmbibliothek!)

```
ENDUPLEX  L ID#E1 304 Bytes
          0123 4567 89AB CDEF ck

000: 54E4 4455 05C4 5485 C7
001: 802E 0032 8191 8009 C0
002: 0420 01E0 1310 0000 59
003: F230 0000 0C60 0000 A3
004: 0DC1 00FF 00BE 000D 31
005: D201 B100 FC10 3010 C1
006: 0DB5 4E44 4550 5420 E4
007: 1954 E444 5505 11D5 A1
008: 4855 4345 5455 431F B6
009: 3545 1425 4555 0542 6B
00A: 211F F31B F961 1131 BC
00B: CF96 1420 0D9D 7320 DB
00C: 0A8F FF81 1321 0A8F 53
00D: FF81 1FC0 0320 0A8F E6
00E: AB81 15FE DBD5 31FC 65
00F: 8FC4 6315 1C31 DC8F 36
010: C463 143B 31DC 8FAF 5B
011: 5311 378D 6262 08DD 23
012: 9730 8D39 4509 FFFF 5B
013: DEFF F8F6 81F0 D231 61
014: 0C10 B320 0A7B 308D D1
015: 84A8 0FCF FF3C FFF8 5F
016: F681 F0D2 CE10 B321 92
017: 0A73 1057 D321 0A8F DD
018: AB81 16D8 F10A 8FE4 79
019: 4A13 1D01 C114 D850 E9
01A: 75B0 8F13 DB01 3710 9E
01B: 8D2E 6E68 B6E0 11A8 6F
01C: F14A 1103 D811 B8BA 9E
01D: 908D A114 1101 11A8 2C
01E: FD79 1149 08DD 4490 75
01F: 1101 3011 98F7 71B1 13
020: 0200 3280 8590 0032 01
021: 00A1 3310 18FA B811 6F
022: 1371 36D5 8F06 4A11 34
023: 1113 15E1 14A3 1D09 15
024: 6221 AE68 F405 8116 E0
025: 156E D913 48D7 B181 CE
```

Hexdump-Listing SHRINKLX

```
SHRINKLX  L ID#E1 229 Bytes
          0123 4567 89AB CDEF ck

000: 3584 2594 E4B4 C485 0E
001: 802E 0055 5172 7009 B4
002: AA10 01E5 6560 0000 21
003: F710 0000 0000 0000 A7
004: 0A30 00DB 3584 2594 C3
005: E4B4 561F F8D9 3850 17
006: 8F5C C305 008D 20F2 74
007: 08EF FFAE FFF8 FD2F E0

008: 9059 08D3 9390 8F36 8B
009: F904 1F13 710B 1358 83
00A: FDF6 7011 2CCC C490 85
00B: 31F3 50D1 13D2 CE10 C7
00C: 9784 0570 8AEA B868 E7
00D: 9031 635E ADBD AD2C EB
00E: E15F 3E65 5017 3137 3B
00F: D5E8 11B8 F10F A1C3 D6
010: 3104 188D 84A8 0131 78
011: 17F1 738F D1F6 0D71 7B
012: 7413 7111 7010 500D A1
013: 2868 0031 6302 101A 79
014: F0CC 1008 4784 8712 4B
015: 0400 1341 10E4 1001 00
016: 198A 2758 7800 1365 F6
017: DDD1 1358 BF44 1731 FC
018: 378B 7E31 5B38 F42A 95
019: 7123 A98D 9B15 2040 A0
01A: 0DD8 1EE6 C6C6 1331 BE
01B: 31C2 8BB5 0858 0385 D9
01C: 702C D858 02         B4
```

SHRINKLX

Hier wieder einmal ein kleiner, aber nützlicher Lexfile, der uns hilft, eine Eigenart (um nicht zu sagen "Bug") des HP-71B Betriebssystems besser zu beherrschen.

Wer viel mit Textfiles arbeitet, dem wird sicher schon aufgefallen sein, daß beim Speichern eines solchen Files auf Mass-Memory (Tape oder Disk, mit Hilfe des COPY-Befehls), die Filelänge stets auf die nächsten vollen 256 Byte aufgerundet wird. Dies hat wohl mit dem Floppy-internen Speichermanagement zu tun und wäre auch nicht weiter schlimm.Leider wird aber beim Zurückladen in den HP-71B dieser zusätzliche, leere Teil des Files mitgeladen, sodaß der File nun im Rechner auch wieder mehr Platz beansprucht als vorher.

Das wäre nun immer noch nicht ganz so schlimm, wenn nicht zusätzlich noch die Neigung des HP-71B wäre, solchen "leeren" Speicherplatz am Ende eines Textfiles bei Vergrößerungen des Files unbenutzt zu lassen! So wird z.B. beim Anfügen eines Textes an das Fileende (z.B. mit dem HP-Texteditor "EDTEXT") die Filegröße um die Anzahl der eingefügten Bytes vergrößert und der ungenutzte Bereich am Ende des Files beibehalten und nach hinten verschoben!

Was sich daraus bei mehrmaligem Laden, Speichern und Bearbeiten eines Textfiles ergibt, kann man sich leicht ausmalen: der File wächst und wächst, weit über seinen tatsächlichen Inhalt hinaus und beansprucht sinnloserweise wertvollen Speicherplatz, sowohl im Rechner, als auch am Speichermedium.

Abhilfe konnte bisher lediglich ein aufwendiges, textorientiertes Umkopieren der betroffenen Files schaffen.

Der französische Kollege Jean-Jacques Moreau hat sich dieses Problems jedoch angenommen und den Lexfile SHRINKLX geschrieben, der übergroße Textfiles auf ihre wahre Größe "zurückschrumpft".

Der File wurde erstmals in JPC 35 vorgestellt, dem Journal des PPC Paris. Die Filegröße ist 229 Bytes (211 laut CAT), es ist 1 Schlüsselwort vorhanden:

```
SHRINKLX  ID=E1    SIZE=229 Bytes
  65 SHRINK  Stmt 225101
*
```

Die korrekte Syntax lautet:

SHRINK *file*

Dabei ist entweder der Filename explizit oder als Stringvariable anzugeben. Falls es sich nicht um einen Textfile handeln sollte, so wird die Fehlermeldung "Invalid File Type" ausgegeben.

Vorher und nachher kann mit CAT leicht die Filegröße und damit der Erfolg von SHRINK überprüft werden. Files ohne "toten" Speicherplatz bleiben unverändert.

Der Vollständigkeit halber sei noch erwähnt, daß es nicht nur mit COPY, sondern auch mit anderen Befehlen möglich ist, toten Speicherplatz am Ende eines Textfiles zu erzeugen. Dazu gehört CREATE TEXT mit einer entsprechenden Größenangabe, das überhaupt einen leeren Textfile erzeugt, und PRINT#, mit dem es möglich ist, am Anfang oder irgendwo mitten im File einen String zu schreiben, an dessen Ende dann das künftige Fileende zu stehen kommt. Alles was dahinter im File war, ist dann "tot" und nicht mehr zugänglich. Also Vorsicht beim Experimentieren!

So wie die meisten anderen Lexfiles der französischen PPC-Autoren, so ist auch der Befehl SHRINK im JPC-Rom enthalten.

Wer mit Hilfe des vorgestellten SHRINK-Befehls nun alle seine Textfiles auf seinen Disketten "gesundschrumpfen" will, sollte darauf achten, daß, nach dem Laden in den Rechner, jeweils der alte File auf Diskette gelöscht werden sollte, bevor der neue, geschrumpfte, mit dem selben Namen zurückgeschrieben wird! Sonst wird eventuell der alte Speicherplatz wiederbenützt, und der Effekt ginge verloren.

Also etwa:

```
COPY TESTTEXT:TAPE
PURGE TESTTEXT:TAPE
SHRINK TESTTEXT
COPY TESTTEXT TO :TAPE
```

Happy Shrinking.

Der Lexfile SHRINKLX ist wie immer entweder durch Abtippen des Listings (mit dem Programm MAKEFILE), oder bei unserer Programmbibliothek (Henry Schimmer, siehe letzte Seite), oder über unsere Mailbox GEO1 zu bekommen.

Michael Fiedler (376)
Friedrichstraße 17
6070 Langen

ThinkJet - Druckersteuerung

PRINTLX ID-71

Eine einfache und schnelle Druckersteuerung kann mit folgenden Befehlen ermöglicht werden. Auch als STARTUP-Version zu empfehlen.

Erforderlich ist das LEX-File "PRINTLX" im Hauptspeicher oder den Ports des HP-71B.

Der Papieranfang muß in der Startposition stehen.

27 BELL
Arbeitet nur bei Druckern die eine Glocke haben (z.B. HP-82905). Theologie-Befehl.

28 BOLD
Schaltet den Fettschrift-Modus ein/aus.
Befehle: BOLD ON, BOLD OFF.
Beispiel:

```
10 BOLD ON @ PRINT "XXX"
                  @ BOLD OFF
```

29 CR
Sendet den Wagenrücklauf-Befehl CHR$(13) an den Drucker.

2A ESC$
Ersetzt den Befehl CHR$(27).

2B FF
Sendet den Seitenvorschub-Befehl CHR$(12) an den Drucker.

2C LF
Sendet den Zeilenvorschub-Befehl CHR$(10) an den Drucker.

2D MODE
Schaltet den Schriftgrößen-Modus des ThinkJet-Druckers ein.

```
MODE0 =  80 Zeichen/Zeile
MODE1 =  40 Zeichen/Zeile
MODE2 = 142 Zeichen/Zeile
MODE3 =  71 Zeichen/Zeile
```

Für andere Drucker gelten die Positionen 0 bis 9.

2E PERF
Schaltet den Persorations-Seitenübersprungs-Befehl ein/aus.

2F PL
Bestimmt die Seitenlänge und die Zeilenzahl.
Befehl für z.B. DIN A4: PL72,66. 72 für max. 72 Zeilen, 66 für gewünschte Zeilen pro Blatt.

30 UNDERLINE
Schaltet den Untersteichungs-Modus des ThinkJet-Druckers ein/aus.
Gilt nur für Drucker (wie ThinkJet), die diesen Modus haben.
Befehle: UNDERLINE ON, UNDERLINE OFF.

31 WRAP
Automatischer Zeilenumbruch. Der übersteeende Text wird in der nächsten Zeile geschrieben.
Befehle: WRAP ON, WRAP OFF.

```
10 MODE 1 @ PRINT 'ThinkJet - Druckersteuerung' @ PRINT @ MODE 0
20 PRINT 'PRINTLX  ID-71' @ PRINT
30 PRINT 'Eine einfache und schnelle Druckersteuerung kann mit den folgenden'
40 PRINT 'Befehlen ermöglicht werden. Auch als STARTUP-Version zu empfehlen.' @
PRINT
50 PRINT 'Erforderlich ist das LEX-FILE  "PRINTLX"  im Hauptspeicher oder in'
60 PRINT 'den Ports des HP-71B.' @ PRINT
70 PRINT 'Der Papieranfang muß in der Startposition stehen.' @ PRINT
80 UNDERLINE ON @ PRINT '27 BELL' @ UNDERLINE OFF
90 PRINT 'Arbeitet nur bei Druckern die eine Glocke haben (z.B. HP-82905).'
100 PRINT 'Theologie-Befehl.' @ PRINT
110 UNDERLINE ON @ PRINT '28 BOLD' @ UNDERLINE OFF
120 PRINT 'Schaltet den Fettschrift-Modus ein / aus.'
130 PRINT 'Befehle: BOLD ON, BOLD OFF.'
140 PRINT 'Beispiel: 10 BOLD ON @ PRINT "XXX" @ BOLD OFF' @ PRINT
150 UNDERLINE ON @ PRINT '29 CR' @ UNDERLINE OFF
160 PRINT 'Sendet den Wagenrücklauf-Befehl CHR$(13) an den Drucker.' @ PRINT
170 UNDERLINE ON @ PRINT '2A ESC$' @ UNDERLINE OFF
180 PRINT 'Ersetzt den Befehl CHR$(27)' @ PRINT
190 UNDERLINE ON @ PRINT '2B FF' @ UNDERLINE OFF
200 PRINT 'Sendet den Seitenvorschub-Befehl CHR$(12) an den Drucker.' @ PRINT
210 UNDERLINE ON @ PRINT '2C LF' @ UNDERLINE OFF
220 PRINT 'Sendet den Zeilenvorschub-Befehl CHR$(10) an den Drucker.' @ PRINT
230 UNDERLINE ON @ PRINT '2D MODE' @ UNDERLINE OFF
240 PRINT 'Schaltet den Schriftgrößen-Modus des ThinkJet-Druckers ein.' @ PRINT
250 PRINT 'MODE0  =  80 Zeichen / Zeile'
260 PRINT 'MODE1  =  40 Zeichen / Zeile'
270 PRINT 'MODE2  = 142 Zeichen / Zeile'
280 PRINT 'MODE3  =  71 Zeichen / Zeile' @ PRINT
290 PRINT 'Für andere Drucker gelten die Positionen 0 bis 9' @ PRINT
300 UNDERLINE ON @ PRINT '2E PERF' @ UNDERLINE OFF
310 PRINT 'Schaltet den Perforations-Seitenübersprungs-Befehl ein / aus.'
320 PRINT 'Befehle: PERF ON, PERF OFF' @ PRINT
330 UNDERLINE ON @ PRINT '2F PL' @ UNDERLINE OFF
340 PRINT 'Bestimmt die Seitenlänge und die Zeilenzahl.'
350 PRINT 'Befehl für z.B. DIN A4:  PL72,66'
360 PRINT '72 für max. 72 Zeilen, 66 für gewünschte Zeilen pro Blatt.' @ PRINT
370 UNDERLINE ON @ PRINT '30 UNDERLIN' @ UNDERLINE OFF
380 PRINT 'Schaltet den Unterstreichungs-Modus des ThinkJet-Druckers ein/aus.'
390 PRINT 'Gilt nur für Drucker (wie ThinkJet) die diesen Modus haben.'
400 PRINT 'Befehle: UNDERLINE ON, UNDERLINE OFF.' @ PRINT
410 UNDERLINE ON @ PRINT '31 WRAP' @ UNDERLINE OFF
420 PRINT 'Automatischer Zeilenumbruch. Der überstehende Text wird in der'
430 PRINT 'nächsten Zeile geschrieben.'
440 PRINT 'Befehle: WRAP ON, WRAP OFF' @ PRINT @ PRINT
450 MODE 2 @ PRINT 'H.A.WUTTKE (CCD 2742)' @ MODE 0
```

H. A. Wuttke (2742)
Hainerweg 271
6000 Frankfurt 70

Infrarot-Thermodrucker HP82240B

Der "alte" Infrarotdrucker wird ab jetzt durch diesen Typ ersetzt (man beachte das "B" am Ende der Nummer), die Lebensdauer der Batterien soll verlängert worden sein, ebenso soll das Druckbild besser geworden sein, dies war aber auch Zeit.

Ebenso wird der Drucker nach 10min. Inaktivität abgeschaltet.

Aktiver Umweltschutz

Hewlett Packard Fachhändler nehmen ab sofort *kostenlos* leere Tonerkassetten der kompletten Laserjet-Familie zurück. Diese werden zu fast 90% wieder in die Produktion zurückgeführt.

Ebenso weist Hewlett Packard darauf hin, daß die leeren Tinten-Patronen der DESKJET-Familie ***kein Sondermüll*** sind, wie oft gemunkelt wurde.

PRISMA Inhaltsverzeichnis

Das Inhaltsverzeichnis von PRISMA ist um die Inhalte der Hefte des Jahres 1989 ergänzt worden und steht allen zur Verfügung (siehe Serviceleistungen).

Wasserfeste Tinte

Ab 1. Januar 1991 will HP für seine DESKJET-Drucker die schon lange angekündigte wasserfeste Tinte anbieten.

MM (1000)

MCODE:
XROM-Funktionen

In der Anfrage aus PRISMA 88-01-37 wird ein Programm gesucht, daß die von einem beliebigen Programm benötigten Einsteckmodule anzeigt.

Nicht zuletzt dank der einfachen Verwendungsmöglichkeiten einiger Betriebssystemroutinen entstand die nun vorliegende Version in MCODE.

Als Modulname wird der erste Funktionsname in der Page angezeigt, in der die betreffende Funktion angesiedelt ist. Der Aufruf von "XROM" erfolgt wie z.B. bei "COPY".

Bei angeschlossenem Drucker wird die Modulliste im Tracemodus ausgedruckt. Ohne Drucker wird immer ein Name angezeigt; nach Betätigung der R/S-Taste der nächste.

Die Funktion ruft bei ungepackten Routinen "PACK" auf, da die sonst evtl. vorhandenen Nullbytes dazu führen, daß einige XROM-Funktionen nicht erkannt werden.

Das Statusregister L wird als Ringpuffer benutzt, in den der signifikante Teil des XROM-Befehls jeweils in einem 1-Byte-Segment gespeichert wird. Es können also maximal 7 verschiedene Modulnamen erfaßt werden.

Das Programm läßt sich in sieben Abschnitte unterteilen:

```
0001-0024: Packen und Programm-
           anfang suchen, der
           sich nur im RAM be-
           finden darf;
0025-0048: XROM-Befehl sequen-
           tiell suchen;
0049-0063: Die zwei mittleren
           Nybbles des XROM-Be-
           fehls zusammenset-
           zen und die zwei
           übriggebliebenen
           Bits der Befehlsnum-
           mer löschen;
0064-0075: Test ob XR-ID schon
           gespeichert wurde;
0076-0089: XR-ID abspeichern;
0090-0141: Anzeige der Modulna-
           men;
0142-0144: Programmende.
```

Ein XROM-Befehl setzt sich wie folgt zusammen:

```
Bits 15-11 = XROM-Kennung
             (10100),
Bits 10- 6 = Peripherie-
             (Modul-) Nummer,
Bits  5- 0 = Funktionsnummer.
```

```
0001    93B4   ØBD ØØF 212 118   "XROM"  erwartet ALPHA-Eingabe (Prgm-Name)
0005    93B8   ØØØ     NOP              Funktion ist nicht programmierbar
0006    93B9   244     CLRF 9           Bedingung für normales 'packen'
0007    93BA   Ø19 ØBØ  ?NCXQ 2ØØ6      'PACKING' ohne Anzeige der Meldung
0009    93BC   1AØ     A=B=C=Ø          klare Anfangsbedingungen schaffen
0010    93BD   128     WRIT 4(L)        hier werden später XR-ID's gespeichert
0011    93BE   278     READ 9(Q)        Programmname nach CPU-C
0012    93BF   1D8     C<>M ALL         Anfangsbedingung für "ASRCH": Alpha-Label
                                        in CPU-M und Register 9(Q)
0013    93CØ   315 Ø98  ?NCXQ 26C5      "ASRCH": Anfangsadresse des Prgm's suchen
0015    93C2   2EE     ?C≠Ø ALL         wenn gefunden, dann Sprung
0016    93C3   381 ØØA  ?NCGO Ø2EØ      "ERRNE": ansonsten Fehlermeldung
0018    93C5   2ØC     ?FSET 2          handelt es sich um eine ROM-Adresse?
0019    93C6   3EF     JC 93C3 -Ø3      wenn ja, dann Fehlermeldung
0020    93C7   ØC4     CLRF 1Ø          Prgm-Zähler auf RAM-Adresse setzen
0021    93C8   ØAE     A<>C ALL         Prgm-Anfangs-Adresse nach CPU-A(3..Ø)
0022    93C9   Ø1C     R= 3             die letzten 3 Befehle sind die Anfangsbedingungen
                                        für "PUTPC", welches die Adresse in PC-Format
0023    93CA   ØDD Ø8C  ?NCXQ 2337      mit Links-Shift von Nybble 3 konvertiert und den
                                        Prgm-Zähler auf den Prgm-Anfang setzt
0025    93CC   141 ØA4  ?NCXQ 295Ø      "GETPC": holt Prgm-Zähler in MM-Format
0027    93CE   ØAE     A<>C ALL         PC nach CPU-C(3..Ø)
0028    93CF   158     M=C ALL          und in CPU-M zwischenspeichern
0029    93DØ   ØAE     A<>C ALL         PC in CPU-A ist Anfangsbedingung für ..
0030    93D1   3E5 ØA8  ?NCXQ 2AF9      "SKPLIN": setzt PC auf nächsten Befehl
0032    93D3   ØDD Ø8C  ?NCXQ 2337      "PUTPC": und speichert ihn ab
0034    93D5   198     C=M ALL          alte Adresse holen
0035    93D6   Ø1C     R= 3             Nybbles 3..Ø auswählen
0036    93D7   36A     ?A≠C R<-         alte und neue Adresse vergleichen
0037    93D8   1AB     JNC 94ØD +35     wenn gleich, dann Prgm-Ende erreicht.
0038    93D9   2E5 ØA4  ?NCXQ 29B9      "NXBYTA": holt Byte nach CPU-C(1..Ø) aus RAM
0040    93DB   ØAE     A<>C ALL         incr. Adresse nach CPU-C
0041    93DC   158     M=C ALL          und in CPU-M zwischenspeichern
0042    93DD   13Ø ØA8  LDI S&X         XROM-Zeile, erstes nicht mehr gültiges Byte
0044    93DF   31C     R= 1             Zeilennummer anwählen (Quick Reference Card)
0045    93EØ   362     ?A≠C àR          könnte es ein XROM-Befehl sein?
```

```
0046   93E1   35F      JC 93CC -15        wenn nein, Sprung zum Schleifenanfang
0047   93E2   3ØA      ?A<C R<-           ist es ein XROM-Befehl ?
0048   93E3   34B      JNC 93CC -17       wenn nicht, dann Sprung zum Schleifenanfang
0049   93E4   ØA6      A<>C S&X           Byte nach CPU-C
0050   93E5   39C      R= Ø               Nybbles Ø und 1 anwählen
0051   93E6   Ø58      G=C àR;+           und nach CPU-G abspeichern
0052   93E7   198      C=M ALL            Adresse nach CPU-C holen
0053   93E8   ØAE      A<>C ALL           und nach CPU-A bringen
0054   93E9   Ø1C      R= 3               letzten 3 Befehle: Anfangsbedingungen
0055   93EA   2E5 ØA4  ?NCXQ 29B9         "NXBYTA": holt nächstes Byte nach CPU-C(1..Ø)
0057   93EC   33C      RCR 1              nur das linke Nybble wird benötigt
0058   93ED   31C      R= 1               Nybbles 1 und 2 anwählen
0059   93EE   Ø98      C=G àR;+           erstes Byte nach CPU-C(2..1)
0060   93EF   ØA6      A<>C S&X           die ersten 3 Nybbles der XR-Nr. nach CPU-A
0061   93FØ   3DØ 31Ø  LDàR F,C          Maske zum isolieren der 5 Bits der XR-ID
0063   93F2   3BØ      C=C AND A          maskieren
0064   93F3   2DC      R= 13              Mantissen-Vorzeichen-Nybble anwählen
0065   93F4   19Ø      LDàR 6             Zähler einstellen, dient zum Test, ob XR-ID
                                           schon gespeichert wurde
0066   93F5   ØAE      A<>C ALL           XR-ID und Zähler nach CPU-A
0067   93F6   Ø46      C=Ø S&X
0068   93F7   27Ø      RAM SLCT
0069   93F8   138      READ 4(L)          XR-ID-Speicher nach CPU-C
0070   93F9   31C      R= 1               Nybbles Ø und 1 anwählen
0071   93FA   36A      ?A≠C R<-           ist XR-ID schon im Register ?
0072   93FB   28B      JNC 93CC -2F       wenn ja, Sprung zum Schleifenanfang
0073   93FC   23C      RCR 2              nächstes Register-Segment
0074   93FD   1BE      A=A-1 MS           Zähler dekrementieren
0075   93FE   3E3      JNC 93FA -Ø4       wenn noch nicht alle Segmente überprüft,
                                           dann Test des nächsten Segments
0076   93FF   ØBE      A<>C MS            ansonsten: XR-ID wurde noch nicht gespeichert
0077   94ØØ   2DC      R= 13
0078   94Ø1   19Ø      LDàR 6             Zähler erneut einstellen
0079   94Ø2   ØBE      A<>C MS            und wieder nach CPU-A
0080   94Ø3   31C      R= 1               Nybbles Ø und 1 anwählen
0081   94Ø4   2EA      ?C≠Ø R<-           ist dieses Register-Segment noch frei ?
0082   94Ø5   Ø2B      JNC 94ØA +Ø5       wenn ja, dann verlasse die Schleife
0083   94Ø6   23C      RCR 2              ansonsten: nächstes Register-Segment
0084   94Ø7   1BE      A=A-1 MS           Zähler dekrementieren
0085   94Ø8   3E3      JNC 94Ø4 -Ø4       Test des nächsten Segments
0086   94Ø9   Ø23      JNC 94ØD +Ø4       Register ist vollständig belegt, es folgt die Anzeige
0087   94ØA   ØAA      A<>C R<-           es wurde ein freies Segment gefunden:
0088   94ØB   128      WRIT 4(L)          die XR-ID wird abgespeichert
0089   94ØC   2BB      JNC 93E3 -29       Sprung in 2 Schritten zum Schleifenanfang
                                           zur Suche der nächsten XR-ID
0090   94ØD   13Ø ØØ7  LDI S&X           Zähler für Ergebnis-Anzeige initialisieren
0092   94ØF   158      M=C ALL            und in CPU-M speichern
0093   941Ø   198      C=M ALL            Anfang der Anzeige-Schleife !
0094   9411   266      C=C-1 S&X          Zähler dekrementieren
0095   9412   18F      JC 9443 +31        nach Anzeige von max. 7 Modulnamen: Prgm-Ende
0096   9413   158      M=C ALL            Zähler wieder nach CPU-M
0097   9414   3C1 ØBØ  ?NCXQ 2CFØ         "CLLCDE": Display anwählen und löschen
0099   9416   149 Ø24  ?NCXQ Ø952         "ENCPØØ": Chip Ø anwählen
0101   9418   138      READ 4(L)
0102   9419   23C      RCR 2              nächstes Segment
0103   941A   128      WRIT 4(L)
0104   941B   Ø56      C=Ø XS             XR-ID isolieren
0105   941C   2E6      ?C≠Ø S&X           war Segment belegt ?
0106   941D   39B      JNC 941Ø -ØD       wenn nicht, Test des nächsten Segments
0107   941E   Ø5E      C=Ø MS
```

```
0108    941F  2FC    RCR 13          rechtes Nybble auf Null setzen
0109    942Ø  Ø1C    R= 3
0110    9421  29Ø    LDàR A          linkes Nybble -> vollständige XR-Nr. (XR xx,ØØ)
0111    9422  149 ØBC ?NCXQ 2F52      "XECROM*": erwartet an diesem Ansprungpunkt
                                      nur die volständige XR-Nr. in CPU-C(3..Ø),
                                      zeigt den Funktionsnamen an
0113    9424  1Ø8    SETF 8          wenn Drucker angeschlossen, soll Ausdruck erfolgen
0114    9425  2Ø1 Ø7Ø ?NCXQ 1C8Ø      "MSG1Ø5": Anzeige (Druck) des Modulnamens
0116    9427  3B8    READ 14(d)      Flagregister
0117    9428  358    ST=C XP         zum testen Flags 48-55 nach CPU-ST
0118    9429  38C    ?FSET Ø         Flag 55 gesetzt, also Drucker angeschlossen ?
0119    942A  Ø33    JNC 943Ø +Ø6    wenn nicht, Sprung zur Tastaturabfrage
0120    942B  Ø46    C=Ø S&X
0121    942C  2A6    C=-C-1 S&X      Pausenschleife initialisieren
0122    942D  266    C=C-1 S&X       so kann man das Ergebnis auch bei angeschlossenem
0123    942E  3FB    JNC 942D -Ø1    Drucker in der Anzeige lesen
0124    942F  Ø7B    JNC 943E +ØF    Sprung zum nächsten Schleifendurchlauf
0125    943Ø  13Ø Ø87 LDI S&X         Tastencode R/S-Taste
0127    9432  ØA6    A<>C S&X        nach CPU-A
0128    9433  3C8    CLRKEY
0129    9434  3CC    ?KEY
0130    9435  3FB    JNC 9434 -Ø1    warten auf Tastendruck
0131    9436  22Ø    C=KEY KY        Tastencode nach CPU-C
0132    9437  Ø3C    RCR 3           ins Exponentenfeld
0133    9438  Ø56    C=Ø XS          das störende 3. Nybble(links) löschen
0134    9439  3C8    CLRKEY          erst im Programm fortfahren, wenn
0135    943A  3CC    ?KEY            die Taste losgelassen wird
0136    943B  3F7    JC 9439 -Ø2
0137    943C  366    ?A≠C S&X        wurde R/S gedrückt ?
0138    943D  Ø27    JC 9441 +Ø4     wenn nicht, Programm beenden
0139    943E  239 ØØC ?NCXQ Ø38E      "RSTMSØ": Message-Flag zurücksetzen
0141    944Ø  283    JNC 941Ø -3Ø    Sprung zum Anfang der Anzeige-Schleife
0142    9441  239 ØØC ?NCXQ Ø38E      "RSTMSØ": Message-Flag zurücksetzen
0144    9443  31D ØØ2 ?NCGO ØØC7      "NFRKB": setzt die Anzeige zurück (?)
```

Klaus Huppertz (3365)
Nievelsteinstraße 30
D-4050 Mönchengladbach 3

Poisson-Verteilung

auf dem HP15C

Diese häufig gebrauchte statistische Verteilungsfunktion

(1) $P(I)=(\exp(-M)*M^I)/I!$

(2) $\sum_{I=0}^{N-1} P(I)=1$

berechnen HP41 und HP15C recht bequem.

Das angegebene Programm löst auch durch Probieren genügend schnell die Aufgabe, mit wieviel Gliedern eine vorgegebene Summe (2) erreicht werden kann. Gewöhnlich genügen drei Dezimalstellen.

Das unten aufgeführte Programm beschreibt die Implementierung auf dem HP15C, dessen Funktion RCL* den Kern der Arbeitsweise sehr erleichtert hat.

Programmschritt	Ein-/ Ausgaben
LBL A	M ENTER N
EEX	
3	
/	
STO I	
X<>Y	
STO 0	
CHS	
E^X	
R/S	P(0)
STO 1	
ISG I	
LBL 0	
RCL* 0	
RCL I	
INT	
/	
R/S	P(I)
ST+ 1	
ISG I	
GTO 0	
RCL 1	
RTN	Σ P(I)+

Dr.G.Heilmann
Obernhofer Straße 15
5408 Seelbach

Poisson-Verteilung auf dem HP41

HP41C, 059 Zeilen,140 Bytes, 20 REGs, SIZE 001 (X-Function),(Printer)

Diese häufig gebrauchte statistische Verteilungsfunktion

(1) $P(I)=(\exp(-M)*M^I)/I!$

(2) $\sum_{I=0}^{N-1} P(I)=1$

berechnen HP41 und HP15C recht bequem.

Das angegebene Programm löst auch durch Probieren genügend schnell die Aufgabe, mit wieviel Gliedern eine vorgegebene Summe (2) erreicht werden kann. Gewöhnlich genügen drei Dezimalstellen.

Das unten aufgeführte Programm beschreibt die Implementierung auf dem HP41 mit X-Function Modul, dessen Funktionen SIZE? und PSIZE allerdings nur der korrekten Anzahl benötigter Speicher dient.
Man kann natürlich auch genügend Daten-Speicher von Hand zur Verfügung stellen, es entfallen dann die Programmzeilen 03-06. Eingestellt wird mit PSIZE bei Bedarf SIZE 001.

```
01*LBL "POISSON"
02 ADV
03 SIZE?
04 1
05 X>Y?
06 PSIZE
07 SF 12
08 SF 21
09 CF 29
10 "POISSON"
11 AVIEW
12 ADV
13 CF 12
14 3
15 "FIX?"
16 PROMPT
17 FIX IND X
18 "ANZ. GLIEDER?"
19 PROMPT
20 DSE X
21  E3
22 /
23 ISG X
24 "M?"
25 PROMPT
26 "M  = "
27 ARCL X
28 AVIEW
29 ADV
30 ENTER^
31 E^X
32 1/X
33 "P0 = "
34 ARCL X
35 AVIEW
36 STO 00
37*LBL 00
38 RCL Y
39 *
40 RCL Z
41 INT
42 /
43 RCL d
44 FIX 0
45 "P"
46 ARCL T
47 "> = "
48 STO d
49 ARCL Y
50 AVIEW
51 RDN
52 ST+ 00
53 ISG Z
54 GTO 00
55 ADV
56 "ΣP = "
57 ARCL 00
58 AVIEW
59 END
```

Dr.G.Heilmann
Obernhofer Straße 15
5408 Seelbach

Tatsächliche Arbeitszeit

HP41C, 44 Zeilen, 083 Bytes, 12 REGs, SIZE 000, TIME-Modul

Gelegentlich weis man gern, welche Zeit für eine bestimmte Arbeit aufgewandt wird, wenn Störungen außer acht bleiben (manchmal ist auch die Summe der Zeiten, in denen die Arbeit nicht fortgesetzt werden kann, viel interessanter).

Anders als der neue HP48SX verfügt der HP41CX über Funktionen, die dieses Problem sehr elegant lösen.
Das HP41-Programm ZT leistet das Gewünschte und benutzt alleine den Stack, also keine Speicherregister.

1. XEQ ZT setzt Flag 01, zeigt die Meldung *"WEITER"*, hält die augenblickliche Zeit fest, löscht Flag 01 und schaltet den Rechner aus.
2. Das nächste ON hält die augenblickliche Zeit fest, meldet *"HALT"*, bildet die Differenz zum vorher festgehaltenen Zeitpunkt und addiert diese Differenz zur bisher berechneten Summe von Zeitintervallen; Flag 01 wird gesetzt und der Rechner schaltet sich wieder aus.
3. Das nächste ON hält die augenblickliche Zeit fest, meldet *"WEITER"*, löscht Flag 01 und der Rechner schaltet sich aus.
4. Das nächste ON führt auf 2. zurück...
5. Halten von R/S und Drücken von ON hält stets den Rechner an, ohne dabei Daten zu zerstören.

XEQ E gibt dann die Summe der bisher addierten Zeitabschnitte im eingestellten Zeitformat aus. Fortsetzung mit R/S, anderes Vorgehen zerstört die Daten.

```
01*LBL "ZT"
02 FIX 4
03 CLST
04 SF 01
05 GTO 00
06*LBL 01
07 SF 11
08*LBL 02
09 OFF
10*LBL 00
11 TIME
12 FS? 01
13 "WEITER"
14 FC? 01
15 "HALT"
16 AVIEW
17 PSE
18 FC? 01
19 XEQ 00
20 FC?C 01
21 SF 01
22 GTO 01
23*LBL 00
24 X<>Y
25 HMS-
26 X<0?
27 XEQ 00
28 HMS+
29 RTN
30*LBL 00
31 24
32 HMS+
33 RTN
34*LBL E
35 SF 11
36 CLA
37 FC? 01
38 X<>Y
39 ATIME
40 FC? 01
41 X<>Y
42 PROMPT
43 GTO 02
44 END
```

Dr.G.Heilmann
Obernhofer Straße 15
5408 Seelbach

Elektronischer Einkaufszettel

148 Zeilen, 307 Bytes, 44 Regs., SIZE 001 + Anzahl Datensätze, HP 41C, X-F, TIME, CCD, ADV, X-I/O

Das nachfolgend beschriebene Programm kann beim Einkaufen helfen, die Übersicht über die anfallenden Kosten zu behalten und dient der Kontrolle des Kassenbons bzw. der Endsumme.

Programmbeschreibung:

Eingabe:
"Stückzahl" ENTER↑ "Preis je Stück"
Beträgt die Stückzahl 1 kann sie entfallen (Defaultwert).

LBL A - Eingabe aufnehmen

LBL B - letzte Eingabe nochmal

LBL C - letzte Eingabe löschen (Rückschritt)

LBL D - Bisherige Eingaben abrufen. Im X-Register zuvor angeben bei der wievielten Einngabe begonnen werden soll. Mit "LBL C" kann die gerade angezeigte Eingabe gelöscht werden.

LBL E - Programm beenden (Die angezeigte Gesamtsumme wird als erster Posten im Datenfile gespeichert. War das Datenfile zu klein gewählt, wird "LBL E" vom Programm selbst angesprungen und in ein "+" an die Gesamtsumme angehängt, um beim späteren Datenaufruf darauf hinzuweisen, daß das Datenfile nicht vollständig ist und ein zweites Datenfile dazugehört. Mit "R/S" kommt man dann weiter zu "LBL a").

LBL a - Programmneustart. Es kann ein neues Datenfile angelegt werden.

LBL b - auf die Frage " DATNR.?" kann ein bestehendes Datenfile zum neuen Arbeitsfile gemacht werden und weiter darin gearbeitet werden.

LBL e - Aufruf um ein Datenfile zu löschen. Auf die Frage " DAT" gibt man die Nummer des zu löschenden Files ein. Anschließend zwei mal "R/S" drücken.

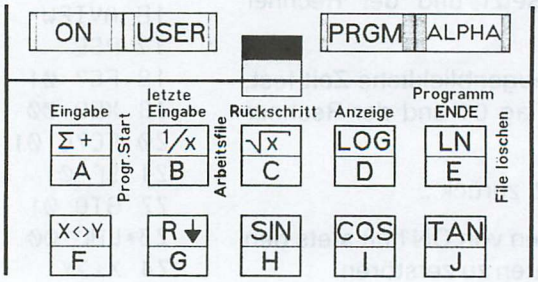
Anmerkungen:

Zu "SIZE?": In 20 Reg. passen 27 Eingaben á "1.99" oder 19 Eingaben á "3" ENTER↑ "1.99"

Für Stückzahl können auch laufende Meter, z.B. für Kabel, eingegeben werden. Die Zwischensumme wird richtig berechnet aber als auf ganze Zahl gerundet abgespeichert. Dieser Posten darf dann nicht gelöscht werden, da "LBL C" mit dem gespeicherten Wert arbeitet und damit von der Gesamtsumme zuviel abzieht. Wer oft mit gebrochenen Stückzahlen arbeitet kann das Format dafür in Zeile 59 ändern. Das kostet dann natürlich Speicherplatz im X-F, läßt dann aber das Löschen zu.

```
01♦LBL "BUY"
02 SF 27
03 CF 28
04 CF 29
05♦LBL a
06 "SIZE?"
07 PROMPT
08 SF 01
09♦LBL b
10 FIX 0
11 "DATNR.?"
12 PROMPT
13 "DAT"
14 ARCL X
15 STO 00
16 X<>Y
17 FS? 01
18 CRFLAS
19 0
20 SEEKPTA
21 "Σ0000,00"
22 FS? 01
23 APPREC
24 FC?C 01
25 GETREC
26 ANUM
27 STO IND 00 DATNR.
28 PROMPT
29 GTO 02
30♦LBL A
31 CLA
32 X<>Y
33 1
34 X=Y?          Stückz.=1?
35 SF 01
36 RDN
37 FS?C 01
38 GTO 03
39 FIX 0
40 ARCL X
41 "⊦*"          "*" anhängen
42♦LBL 03
43 FIX 2
44 ARCL Y
45 STO T
46 X<>Y
47 STO Z
48 SF 25
49 APPREC
                 File zu klein?
50 FC? 25
51 SF 02
52 FC?C 25
53 GTO E
54 *
55 "="
56 ARCL X
57 ST+ IND 00
58 "⊦ Σ"
59 ARCL IND 00
                 Default
60 1
61 PROMPT
62 GTO 02
63♦LBL B
64 RDN
65 RDN
66 GTO A
67♦LBL C
68 RCLPT
69 INT
70 SEEKPT
71 GETREC
                 42="*"
72 42
73 POSA
74 -1
                 keine Stückz.?
75 X=Y?
76 SF 01
77 "⊦*"
78 ANUM
79 ENTER↑
80 42
81 POSA
82 AROT
83 RDN
                 Preis
84 ANUM
                 Summe
85 *
86 FS?C 01
87 ANUM
                 von Σ abziehen
88 ST- IND 00
89 RCLPT
90 INT
91 SEEKPT
                 Posten löschen
92 DELREC
93♦LBL 02
94 FIX 2
                 Default
95 1
96 "      Σ"
97 ARCL IND 00
98 PROMPT
99 CLA
100 GTO 02
101♦LBL D
102 SEEKPT
103 FIX 0
```

Bild: Tastenbelegung zu "BUY"



```
104 "DAT"
105 ARCL 00
106 "├ ab "
107 ARCL X
108 PROMPT
109*LBL 01
110 SF 25
111 GETREC
112 "├  "
113 RCLPT
114 INT
115 ARCL X
116 "├."
117 FC?C 25
118 GTO 02
119 1          Default
120 PROMPT
121 GTO 01
122*LBL e
123 FIX 0
124 "DAT"
125 PROMPT
126 ARCL X
127 PROMPT
128 PURFL
129 STOP       mit "R/S"
130 SF 70      nicht weiter
131*LBL E
132 FIX 0
133 "END"
134 ARCL 00
135 BEEP
136 PROMPT
137 FIX 2
138 0
139 SEEKPT
140 DELREC
141 "Σ"
142 ARCL IND 00
143 FS?C 02        wenn File zu klein
144 "├+"
145 INSREC
146 PROMPT
147 GTO a
148 END
```

Wolfgang Führer
Sachsenstraße 166
D-4350 Recklinghausen

Auflistung der Bytes eines ASCII-Files

31 Zeilen, 248 Bytes, 36 Regs., SIZE 000, HP-41 CX, CCD, HP-IL, DeskJet / ThinkJet

LBL "a↑B" (Alpha zu Bytes dezimal)

Um Übersicht über einen ASCII-File zu erhalten, s. #KEYS, ist es nötig die einzelnen Bytes zu sehen, da das Display des HP-41 in dieser Hinsicht über viele der Zeichen keine Auskunft gibt.

Der Balken nach dem 14. Byte (die Zählung fängt ja bei 0 an) oben und unten am Ausdruck zeigt an, daß hier eine Alpha-Programmzeile zu Ende ist, die mit |- auf eine Länge von 24 ergänzt werden kann. Das Programm ist nur für 1 EMDIRX eingerichtet.

Durch Veränderung der Steuerzeichen ist dieses Programm auch für den ThinkJet-Drucker geeignet. Die Zeilen bis LBL 01 lauten dann

```
01*LBL "a↑B"
02 FIX 0
03 CF 28
04 CF 29
05 "⊕&k1S"
06 ACA
07  E
08 EMDIRX
09 "┤⊕&k2S"
10 OUTA
11 66,032
12 ACLX
13 "⊕⊕=┤⊕&dD"
14 SF 17
15 OUTA
16 CF 17
17 ,023
18 STO 00
19 6,032
20 ACLX
21 4
22*LBL 00
23 RCL 00
24 INT
25 CLA
26 ACAXY
27 ISG 00
28 GTO 00
29 PRBUF
30 "⊕&d@⊕"
31 ACA
32 ,
33 SEEKPT
34 SF 25
35*LBL 01
```

```
005: F5 1B 26 6B 31 53
009: F6 7F 1B 26 6B 32 53
013: F8 FC 1B 3D 0D 1B 26 64 44
030: F5 1B 26 64 40 0A
```

```
01*LBL "a↑B"
02 FIX 0
03 CF 17
04 CF 28
05  E
06 SELECT
07 "⊕1⊕!⊕(10U"
08 ACA
09 "DRAFT:LET? AE"
10 PMTK
11 "⊕(s"
12 ARCL X
13 "┤q5H⊕&17e⊕&a4L"
14 ACA
15 X<>Y
16 EMDIRX
17 "┤⊕⊕⊕(s15H⊕&a14L"
18 OUTA
19 66,032
20 ACLX
21 "⊕⊕=┤⊕&d1D"
22 SF 17
23 OUTA
24 CF 17
25 ,023
26 STO 00
27 6,032
28 ACLX
29 4
30*LBL 00
31 RCL 00
32 INT
33 CLA
34 ACAXY
35 ISG 00
36 GTO 00
37 PRBUF
38 "⊕&d@⊕"
39 ACA
40 ,
41 SEEKPT
42 SF 25
43*LBL 01
44 GETREC
45 FC? 25
46 GTO 05
47 3
48 RCLPT
49 INT
50 SEEKPT
51 "⊕="
52 ACA
53 CLA
54 ACAXY
55 "┤..."
56 ACA
57 GETREC
58*LBL 02
59  E2
60 ATOXL
61 X<0?
62 GTO 03
63 X<Y?
64 XEQ 04
65  E
66 X<>Y
67 X<Y?
68 XEQ 04
69 ACX
70 GTO 02
71*LBL 03
72 PRBUF
73 FC? 17
74 GTO 01
75 6,032
76 ACLX
77 GETREC
78 GTO 02
79*LBL 04
80 32
81 ACCHR
82 X<>Y
83 RTN
84*LBL 05
85 66,032
86 ACLX
87 "⊕"
88 OUTA
89 "⊕E"
90 ACA
91 END
```

Nicht vergessen, in der neuen Zeile 079, das alte Byte durch das Byte 252 zu ersetzen!

Da bei einem über ein Interface angeschlossenen Drucker das IL-Modul keinen Drucker finden kann, muß das Interface mit der angenommenen Adresse 1 (Zeile 5) mittels des Befehls SELECT "manuell" als primäre Zieladresse festgelegt werden, dorthin gehen dann alle Druckbefehle wie ACA oder ähnliche.

```
007: F9 1B 31 1B 21 1B 28 31 30 55
011: F3 1B 28 73
013: FE 7F 71 35 48 1B 26 6C 37 65 1B 26 61 34 4C
017: FF 7F 0A 0A 1B 28 73 31 35 48 1B 26 61 31 34 4C
021: F9 DD 1B 3D 0D 1B 26 64 31 44
038: F5 1B 26 64 40 0A
051: F2 1B 3D
087: F1 DD
089: F2 1B 45
```

Dr. Martin Hochenegger
Heidelberger Landstr. 97
6100 Darmstadt 13

Tastenfeldschablonen auf dem DeskJet

121 Zeilen, 1084 Bytes, 155 Regs., SIZE 000, HP-41 CX, IL- / paralles oder serielles Interface, DeskJet

Die von HP mitgelieferten Schablonen für die Tastenbelegungen sind zum Beschriften bei einer Doppelbelegung zu klein. Die zweite Beschriftung kann nur senkrecht neben der ersten erfolgen. Die Benützung von zwei Schablonen ist natürlich möglich.

Zuerst verwendete ich einen Fotowürfel. Das Tastenfeld wurde auf Papier auf- und die Belegungen eingetragen. Ich hatte so sechs Möglichkeiten.

Meine zweite Variante war die Schablone der Abb. 1. Man kann nach PRKEYS am IL-Drucker die Befehle ausschneiden und in die Leerfelder (nur mit Pinzette möglich) einkleben. Nicht vergessen, das Blatt zu kopieren! Dieses wird auf dünnen Karton (eventuell mit Doppelklebefolie) aufgeklebt (Abb. 2).

Die Schablone, Maske der Abb. 3, wurde mittels eines DeskJet-Druckers hergestellt. Die Leerräume werden mit den Tastenbelegungen im ED-Modus, Ersetzungskursor, ausgefüllt.

Die Schablone ist aus den Linienzeichen des PC8-Zeichensatzes zusammengesetzt (179_{10} = "|" senkrechter Strich). Anstelle der Leerzeichen kann man Buchstaben einsetzen (die Leerzeichen haben den Dezimalwert 32), die den Text der Tastenbelegung darstellen.

Um das mühevolle Eingeben dieser Maske zu ersparen, ist hier ein Programm abgedruckt. Es ist keineswegs optimal, erfüllt aber seinen Zweck, zumal es ja nur einmal benötigt wird um den File zu erstellen.

Das Programm läßt sich am IL-Drucker nicht ausgeben, da manche Bytes der 2. Spalte der Byte-Tabelle den Ausdruck verändern (vergleiche mein Programm "Zeitanzeige auf PAC-SCREEN" in PRISMA 88-6-34).

Zum Ausdruck des Rahmens dient folgendes kleines Programm:

```
01*LBL "↑"
02 ,
03 SEEKPT
04*LBL 00
05 GETREC
06 OUTA
07 GTO 00
08 .END.
```

```
01*LBL "$KEYS"
02 "#KEYS"
03 127
04 CRFLAS
05 "@1@!@(s2q5H    "
06 "}@&d2D#KEY"
07 APPREC
08 "S@&d@"
09 APPCHR
10 "@(1OU@(s15H@&k0"
11 "}W@@&l8C"
12 APPREC
13 "@@@@@@@@@@@@@@@@"
14 "}@@@@@@@@@"
15 APPREC
16 "@@@@@@@@@@@@@@@@"
17 "}@@"
18 APPCHR
19 "@&d1D@       @ "
20 "}      @  "
21 APPREC
22 "     @       @ "
23 "}      @"
24 APPCHR
25 "@&d2D@       @ "
26 "}      @  "
27 APPREC
28 "     @       @ "
29 "}      @"
30 APPCHR
31 "@&d1D@       @ "
32 "}      @  "
33 APPREC
34 "     @       @ "
35 "}      @"
36 APPCHR
37 "@&d2D@       @ "
38 "}      @  "
39 APPREC
40 "     @       @ "
41 "}      @"
42 APPCHR
43 "@&d1D@       @ "
44 "}      @  "
45 APPREC
46 "     @       @ "
47 "}      @"
48 APPCHR
49 "@&d2D@       @ "
50 "}      @  "
51 APPREC
52 "     @       @ "
53 "}      @"
54 APPCHR
55 "@&d1D@       @ "
56 "}      @  "
57 APPREC
58 "     @       @ "
59 "}      @"
60 APPCHR
61 "@&d2D@       @ "
62 "}      @  "
63 APPREC
64 "     @       @ "
65 "}      @"
66 APPCHR
67 "@&d1D@       @ "
68 "}@       "
69 APPREC
70 " @       @   "
71 "}      @"
72 APPCHR
73 "@&d2D@       @ "
74 "}@       "
75 APPREC
76 " @       @   "
77 "}      @"
78 APPCHR
79 "@&d1D@       @ "
80 "}@       "
81 APPREC
82 " @       @   "
83 "}      @"
84 APPCHR
85 "@&d2D@       @ "
86 "}@       "
87 APPREC
88 " @       @   "
89 "}      @"
90 APPCHR
91 "@&d1D@       @ "
92 "}@       "
93 APPREC
94 " @       @   "
95 "}      @"
96 APPCHR
97 "@&d2D@       @ "
```

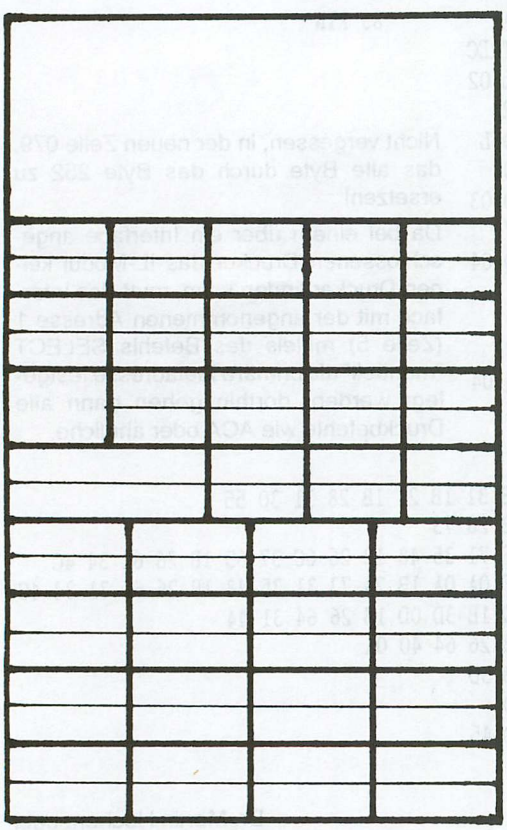


Abb. 1

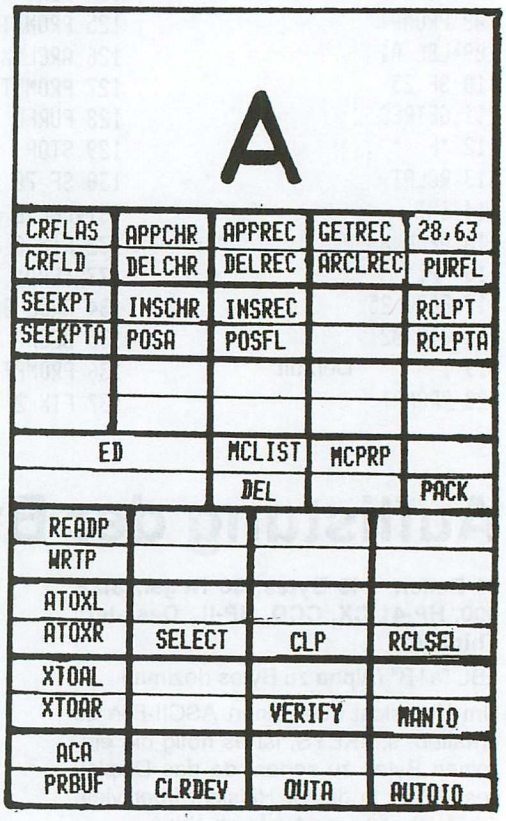


Abb. 2

#KEYS

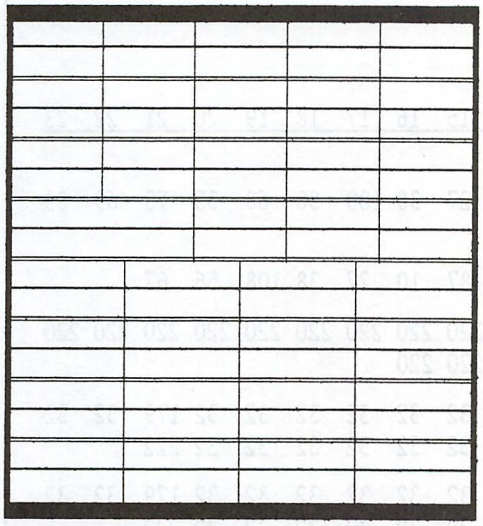


Abb. 3

```
 98 "}⊖          "
 99 APPREC
100 " ⊖          ⊖   "
101 "}       ⊖"
102 APPCHR
103 "⊖&d1D⊖          "
104 "}⊖          "
105 APPREC
106 " ⊖          ⊖   "
107 "}       ⊖"
108 APPCHR
109 "⊖⊖&d@          ⊖"
110 "}          "
111 APPREC
112 "⊖          ⊖   "
113 "}       ⊖"
114 APPCHR
115 "⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖"
116 "}⊖⊖⊖⊖⊖⊖⊖⊖⊖"
117 APPREC
118 "⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖⊖"
119 "}⊖⊖"
120 APPCHR
121 END
```

A

ATOBUFX	APPCHR	APPREC	GETREC	28.63
SACA	DELCHR	DELREC	ARCLREC	PURFL
SEEKPT	INSCHR	INSREC	READXM	RCLPT
SEEKPTA	POSA	POSFL	WRTXM	RCLPTA
ED		MCLST	MCPRP	
		DEL		PACK

READP			
WRTP			
ATOXL			
ATOXR	SELECT	CLP	RCLSEL
XTOAL			
XTOAR		VERIFY	MANIO
ACA			
PRBUF	CLRDEV	OUTA	AUTOIO

Abb. 4

```
005: FF 1B 31 1B 21 1B 28 73 32 71 35 48 20 20 20 20
006: FA 7F 1B 26 64 32 44 23 4B 45 59
008: F5 53 1B 26 64 40
010: FF 1B 28 31 30 55 1B 28 73 31 35 48 1B 26 6B 30
011: F8 7F 57 0A 1B 26 6C 38 43
013: FF DC DC DC DC DC DC DC DC DC DC DC DC DC DC DC
014: FA 7F DC DC DC DC DC DC DC DC
016: FF DC DC DC DC DC DC DC DC DC DC DC DC DC DC DC
017: F3 7F DC DC
019: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 B3 20
020: FA 7F 20 20 20 20 20 20 B3 20 20
022: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
023: F8 7F 20 20 20 20 20 20 DE
025: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 B3 20
026: FA 7F 20 20 20 20 20 20 B3 20 20
028: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
029: F8 7F 20 20 20 20 20 20 DE
031: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 B3 20
032: FA 7F 20 20 20 20 20 20 B3 20 20
034: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
035: F8 7F 20 20 20 20 20 20 DE
037: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 B3 20
038: FA 7F 20 20 20 20 20 20 B3 20 20
040: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
041: F8 7F 20 20 20 20 20 20 DE
043: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 B3 20
044: FA 7F 20 20 20 20 20 20 B3 20 20
046: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
047: F8 7F 20 20 20 20 20 20 DE
049: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 B3 20
050: FA 7F 20 20 20 20 20 20 B3 20 20
052: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
053: F8 7F 20 20 20 20 20 20 DE
055: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 20 20
056: FA 7F 20 20 20 20 20 20 B3 20 20
058: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
059: F8 7F 20 20 20 20 20 20 DE
061: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 20 20
062: FA 7F 20 20 20 20 20 20 B3 20 20
064: FF 20 20 20 20 20 B3 20 20 20 20 20 20 20 B3 20
065: F8 7F 20 20 20 20 20 20 DE
067: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 20 20
068: FA 7F B3 20 20 20 20 20 20 20
070: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
071: F8 7F 20 20 20 20 20 20 DE
073: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 20 20
074: FA 7F B3 20 20 20 20 20 20 20
076: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
077: F8 7F 20 20 20 20 20 20 DE
079: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 20 20
080: FA 7F B3 20 20 20 20 20 20 20
082: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
083: F8 7F 20 20 20 20 20 20 DE
085: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 20 20
086: FA 7F B3 20 20 20 20 20 20 20
088: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
089: F8 7F 20 20 20 20 20 20 DE
091: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 20 20
092: FA 7F B3 20 20 20 20 20 20 20
094: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
095: F8 7F 20 20 20 20 20 20 DE
097: FF 1B 26 64 32 44 DD 20 20 20 20 20 20 20 20 20
098: FA 7F B3 20 20 20 20 20 20 20
100: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
101: F8 7F 20 20 20 20 20 20 DE
103: FF 1B 26 64 31 44 DD 20 20 20 20 20 20 20 20 20
104: FA 7F B3 20 20 20 20 20 20 20
106: FF 20 B3 20 20 20 20 20 20 20 20 20 B3 20 20 20
107: F8 7F 20 20 20 20 20 20 DE
109: FF DD 1B 26 64 40 20 20 20 20 20 20 20 20 20 B3
112: FF B3 20 20 20 20 20 20 20 20 20 B3 20 20 20 20
113: F7 7F 20 20 20 20 20 DE
115: FF DF DF DF DF DF DF DF DF DF DF DF DF DF DF DF
116: FA 7F DF DF DF DF DF DF DF DF
118: FF DF DF DF DF DF DF DF DF DF DF DF DF DF DF DF
119: F3 7F DF DF
```

Tabelle 1:
Textzeilen im Programm $KEYS

#KEYS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0...	27	49	27	33	27	40	115	50	113	53	72	32	32	32	32	27	38	100	50	68	35	75	69	89
	83	27	38	100	64																			
1...	27	40	49	48	85	27	40	115	49	53	72	27	38	107	48	87	10	27	38	108	56	67		
2...	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220
	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220	220								
3...	27	38	100	49	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
4...	27	38	100	50	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
5...	27	38	100	49	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
6...	27	38	100	50	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
7...	27	38	100	49	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
8...	27	38	100	50	68	221	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
9...	27	38	100	49	68	221	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
10...	27	38	100	50	68	221	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	179	32	32
	32	32	32	32	32	179	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	222		
11...	27	38	100	49	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
12...	27	38	100	50	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
13...	27	38	100	49	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
14...	27	38	100	50	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
15...	27	38	100	49	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
16...	27	38	100	50	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
17...	27	38	100	49	68	221	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32
	32	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	222		
18...	221	27	38	100	64	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32
	179	32	32	32	32	32	32	32	32	32	179	32	32	32	32	32	32	32	32	32	32	222		
19...	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223
	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223	223							

Tabelle 2:
Textfile #KEYS ohne Tastenbelegungen

Dr. Martin Hochenegger
Heidelberger Landstraße 97
D-6100 Darmstadt 13

Komfortable
Listenverarbeitung auf dem HP-48 SX

von Matthias Rabe

In dem Namen der Programmiersprache des HP-48, "RPL" steht das "L" für "list processing". Sieht man sich die vielen Funktionen des HP-48 einmal genauer an, so muß man leider feststellen, daß es keine besonders leistungsfähigen Funktionen zur Listenmanipulation gibt. Dem kann abgeholfen werden: Hiermit stelle ich ein paar Funktionen vor, mit denen auch komplexe Operationen oft erstaunlich einfach ausgedrückt werden können.

Die meisten Funktionen sind alles andere als übersichtlich und leicht verständlich geschrieben. Ich lasse weitestgehend alle Funktionen auf dem Stack operieren, wo dann wirklich kaum noch jemand durchblickt. Allerdings lassen sich dadurch teilweise gewaltige Geschwindigkeitssteigerungen erzielen. So habe ich z.B. TRANSPOSE zuerst mit PUT und GET arbeiten lassen. Das war schrecklich langsam. Die jetzige Version ist mehr als zehn mal so schnell. Ich denke, das ist Argument genug, um diese Stackrödelei auf sich zu nehmen.

Im folgenden bedeutet "Funktion" immer ein Objekt, das, wenn es ausgewertet (EVALuiert) wird, einen oder mehrere Parameter vom Stack nimmt, irgendetwas damit macht und dann ein Objekt wieder zurückgibt. Das kann z.B. eine eingebaute Funktion wie SIN sein, oder ein Programm, ein Variablenname, der ein Programm enthält, eine Liste mit einem oder mehreren solchen Objekten (EVAL auf eine Liste angewand, führt die in ihr enthaltenen Objekte aus. So ergibt "1 2 {+} EVAL" 3) etc..

Prädikat steht im Folgenden für eine Funktion (im obigen Sinn), die ein Flag (einen numerischen Wert von Null (= FALSE) oder ungleich Null (= TRUE)) zurückgibt.

Bei fast allen Routinen habe ich auf Fehlerabfragen verzichtet. Mit ungeeigneten Parametern hinterlassen diese Funktionen dann einen Haufen Datenmüll auf dem Stack. Mir erschien Geschwindigkeit wichtiger, als eine genaue Kontrolle der Parameter.

Es muß darauf geachtet werden, daß die Funktionen, die einigen Routinen übergeben werden, genau die richtige Anzahl von Parametern vom Stack nehmen und genau ein Objekt wieder zurückgeben. Sonst gibt es das große Chaos!!!

MAP wendet eine Funktion auf jedes Element einer Liste an.

Liste
Funktion → Liste

Bsp.: { 1 2 Name "String" { } }
{ →STR }
→
{ "1" "2" "'Name'" "String" "{ }" }

MAP2 wendet eine Funktion auf die Elemente zweier Listen an. Es werden jeweils die ersten Elemente beider Listen verknüpft, dann die zweiten etc. und eine Liste dieser resultierenden Elemente zurückgegeben, wobei diese Liste die Länge der kürzeren übergebenen hat.

Liste1
Liste2
Funktion → Liste

Bsp.: { 1 2 3 4 5 }
{ 1 2 3 4 5 6 7 }
{ + }
→
{ 2 4 6 8 10 }

FILTER wendet ein Prädikat auf jedes Element einer Liste an und entfernt alle Elemente, bei denen sich ein FALSE ergibt.

Liste
Prädikat → Liste

Bsp.: { 1 2 3 4 5 6 7 8 9 10 }
{ 2 MOD }
→
{ 1 3 5 7 9}

SPLIT wie FILTER, jedoch werden zwei Listen zurückgegeben. Eine mit allen Elementen, bei denen sich TRUE ergab und eine mit den Elementen, bei denen sich FALSE ergab.

Liste Liste
Prädikat → Liste

Bsp.: { 1 2 3 4 5 6 7 8 9 10 }
{ 2 MOD }
→
{ 1 3 5 7 9 }
{ 2 4 6 8 10 }

FOLDL1 wendet eine Funktion zwischen allen Elementen einer nicht leeren Liste an. Enthält eine Liste nur ein Element, so wird dieses zurückgegeben. Besteht eine Liste aus mehr als zwei Elementen, sagen wir n, so wird $((((e^1 \bullet e^2) \bullet e^3) \ldots) \bullet e^n)$ ausgewertet, wobei "•" die gewünschte Operation ist. In Funktionaler Schreibweise lautet das:
$f(\ldots f(f(f(e^1, e^2), e^3), e^4) \ldots, e^n)$,
wobei f die Funktion ist.

Liste
Funktion → Objekt

Bsp.: { 32 5 75 4 0 -43 }
{ MAX }
→
75
oder: { 1 2 3 4 5 6 }
{ * }
→
720 (= 6!)
oder: { 1 2 "hallo" 3 4 }
{ + }
→
"3hallo34"

FOLDR1 wie FOLDL1, jedoch wird von rechts geklammert. Bei nichtleeren Listen also:
$(e^1 \bullet (e^2 \bullet (e^3 \bullet (\ldots \bullet e^n))))$.
Ist die Funktion kommutativ, also die Klammerung egal, so ist FOLDR1 FOLDL1 vorzuziehen, da es etwas schneller ist.

Liste
Funktion → Objekt

Bsp.: { 1 2 "hallo" 3 4 }
{ + }
→
"12hallo7"

FOLDL arbeitet wie FOLDL1, mit dem Unterschied, daß ein weiteres Objekt angegeben werden muß, mit dem zuerst das erste Listenelement verknüpft wird. Für nichtleere Listen lautet das Ergebnis dann:
$((((obj \bullet e^1) \bullet e^2) \ldots) \bullet e^n)$

Liste
Objekt
Funktion → Objekt

Bsp.: { "hallo" "du" "da" }
0
{ SIZE MAX }
→
5

FOLDR wie FOLDL, jedoch umgekehrte Klammerung, also:
$(e^1 \bullet (e^2 \bullet (\ldots \bullet (e^n \ obj))))$

Liste
Objekt
Funktion → Objekt

Bsp.: { 1 2 "hallo" 3 4 }
""
{ + }
→
"12hallo34"

FOLDLN wendet bei einer Liste von Listen FOLDL1 auf die ersten

Elemente aller Listen an, dann auf die zweiten etc..
MAP2 entspricht
« 2 →LIST FOLDLN »

Liste_von_Listen
Funktion → einfache_Liste

Bsp.: {{ 1 2 3 }{ " hallo" " du" " da" }}
{ + }
→
{ "1 hallo" "2 du" "3 da" }

FOLDRN wendet bei einer Liste von Listen FOLDR1 auf die entsprechenden Elemente aller Listen an.

Liste_von_Listen
Funktion → einfache_Liste

Bsp.: {{ 2 5 54 }{ 5 32 1 }{ -5 3 75 }}
{ MAX }
→
{ 5 32 75 }

SCAN nimmt ein Objekt, eine Liste und eine Funktion vom Stack und wendet die Funktion zuerst auf das Objekt und das erste Listenelement an, dann auf dieses Ergebnis und das nächste Listenelement etc. und gibt dann eine Liste mit all diesen Zwischenergebnissen zurück.

Objekt
Liste
Funktion → Liste

Bsp.: 0
{ 31 29 31 30 31 30 31 31 30 31 30 31 }
{ + }
→
{ 0 31 60 91 121 152 182 213 244 274 305 335 366 }

Diese Liste gibt für jeden ersten eines Monats in einem Schaltjahr an, wie alt das Jahr schon ist, bzw an dreizehnter Stelle die Länge des Jahres.

SCAN1 wie SCAN, jedoch nur für nichtleere Listen.

Liste
Funktion → Liste

Bsp.: { 1 2 3 4 5 }
{ * }
→
{ 1 2 6 24 120 }

Dies ist eine Liste der Fakultäten von eins bis fünf.

LINES nimmt einen String und gibt eine Liste von Strings zurück, die aus den einzelnen Zeilen des übergebenen Strings bestehen.

String → Liste

Bsp.: "Zeile •1Zeile • 2Zeile 3"
→
{ "Zeile 1" "Zeile 2" "Zeile 3" }

"•" in obiger Zeile steht für die Zeilenschaltung (carriage return).

LAY ist die inverse Funktion zu LINES. Sie nimmt eine Liste von Strings (oder anderen Objekten, die dann in Strings umgewandelt werden) und gibt einen String zurück, bei dem die alten Listenelemente durch Zeilenschaltung getrennt sind.

Liste → String

Bsp.: { "Hallo" "du" "da!" }
→
"Hallo•du•da!"

"•" in obiger Zeile steht für die Zeilenschaltung (carriage return).

CONCAT nimmt eine Liste von Listen und verbindet diese zu einer Liste. Ist ein Element keine Liste, so wird es einfach übernommen.

Liste_von_Listen → Liste

Bsp.: {{ 1 2 3 }{ 4 { 5 6 }} 7 8 { 9 10 }}
→
{ 1 2 3 4 { 5 6 } 7 8 9 10 }

CONCAT2 Verbindet zwei Listen zu einer. Handelt es sich bei einem oder beiden Objekten nicht um Listen, so werden sie einfach übernommen.

Liste
Liste → Liste

Bsp.: { 1 { "text" 654 } "mehr Text" }
{ "Listenelement" { 1 2 3 } }

{ 1 { "text" 645 } "mehr Text"
"Listenelement" { 1 2 3 }}

PREFIX fügt ein Objekt an den Anfang einer Liste an.

Objekt
Liste Liste

Bsp.: { "Dies ist ein Listenelement" }
{ "dieses auch" }
→
{{ "Dies ist ein Listenelement" }
"dieses auch" }

SUFFIX fügt ein Objekt an das Ende einer Liste an.

Liste
Objekt → Liste

Bsp.: { 1 2 3 }
« DUP + * »
→
{ 1 2 3 « DUP + * » }

HD gibt das erste Element eine Liste zurück. Dies entspricht « 1 GET » , jedoch wird dabei intern immer nur ein Zeiger auf dieses Element zurückgegeben, was in den meisten Fällen nicht schlimm ist. Jedoch bleibt, auch wenn der Rest der Liste nicht mehr gebraucht wird, bei der Garbage Collection, also wenn der Speicher gepackt wird, der Speicherplatz der ganzen Liste reserviert, wodurch man gelegentlich in völlig unverständliche Speicherenge geraten kann. Ich würde dieses Verhalten als Bug bezeichnen. Dieses Fehlverhalten ist mir beim Testen einiger Sortieralgorithmen aufgefallen, wo plötzlich 20kB nicht mehr ausreichten, um eine Liste mit 100 Zahlen zu sortieren. HD extrahiert nun eine Subliste mit nur einem Element und GETted dieses dann daraus. Dadurch entsteht, unabhängig von der Listenlänge ein Overhead von nur fünf Bytes.

Liste → Objekt

Bsp.: { 1 2 3 }
→
1

TL (tail) gibt eine Liste ohne ihr erstes Element zurück.

Liste → Liste

Bsp.: { 1 2 3 }
→
{ 2 3 }

INIT gibt eine Liste ohne ihr letztes Element zurück.

Liste → Liste

Bsp.: { 1 2 3 }
→
{ 1 2 }

LST (last) gibt das letzte Element einer Liste zurück. Siehe Bemerkung zu HD.

Liste → Objekt

Bsp.: { 1 2 3 }
→
3

DROPWHILE entfernt soviele Elemente aus einer Liste, bis das Prädikat FALSE auswertet.

Liste → Liste

Bsp.: { 2 3 5 3 1 54 6 3 42 31 2 }
{ 10 }
→
{ 54 6 3 42 31 2 }

TAKEWHILE übernimmt soviele Elemente aus einer Liste, bis das Prädikat FALSE auswertet.

Liste
Num → Liste

Bsp.: { 2 3 5 3 1 54 6 3 42 31 2 }
→
{ 2 3 5 3 1 }

RPT (repeat) wiederholt eine Liste

stand in EI abgelegt. Die folgende Batchdatei gibt 10mal "Zonk !" aus.

```
ECHO OFF                    |
BATCHMAN SETLOOP 10         | Zähler=10
:LOOP                       | Sprungmarke Schleifenanf.
BATCHMAN CECHO Zonk !       | "Zonk !" ausgeben
BATCHMAN DECLOOP            | Zähler = Zähler - 1
IF ERRORLEVEL 1 GOTO LOOP
                            | Wenn Zähler >= 1 gehe
                                nach LOOP
```

SETCURSOR x,y {EI=0}
Setzt den Cursor in die x-te Zeile und die y-te Spalte.

TYPEMATIC [m,n | N]
{EI=1 wenn ungültige Parameter, sonst EI=0}
Stellt die Wiederholungsrate der Tastatur ein.
m = Wiederholungrate (0 - 31), wobei größeres m = schnellere Rate
n = Anfangsverzögerung (0 - 3), wobei größeres n = längere Verzögerung
Wird N statt m,n angegeben wird die Wiederholungrate auf m=20 und n=1 eingestellt, wird kein Parameter angegeben wird m=25 und n=0 eingestellt.

VIDEOMODE {EI=Videomodus (0-19)}
Ermittelt den Videomodus.

WAITFOR [mm:]ss {EI=1 wenn abgebrochen, sonst EI=0}
Wartet die angegebene Zeit. Kann mit einem Tastendruck abgebrochen werden.

WAITTIL hh:mm[:ss] {EI=1 wenn abgebrochen, sonst EI=0}
Wertet bis zur angegebenen Zeit. Kann mit einer Taste abgebrochen werden.

WINDOW x,y,b,h,[f,r] {EI=1 bei Fehler, sonst EI=0}
WINDOW erzeugt ein Fenster auf dem Bildschirm. x,y sind die Zeile, Spalte der linken oberen Ecke des Fensters, b die Breite und h die Höhe des Fesnters. Mit f kann eine Farbe für das Fenster an- gegeben werden, mit r kann ein Rahmen erzeugt werden und zwar steht ein Minus "-" für einen einfachen und ein Gleich "=" für einen doppelten Rahmen.

YEAR {EI=Jahr-1980 (0-199)}
MONTH {EI=Monat (1-12)}
DAY {EI=Tag (1-31)}
WEEKDAY {EI=Wochentag (Sontag=0; Samstag=6)}
HOUR {EI=Stunden (0-23)}
MINUTE {EI=Minuten (0-59)}
SECOND {EI=Sekunden (0-59)}
Mit diesen Befehlen können Systemdatum und Zeit ermittelt werden.

Quelle:

PC-Magazine, New York, Volume 9 Number 2

CCD e.V. BATCHMAN

Funktion:

BATCHMAN ist ein Utility, das Batchdateien eine Sammlung von 48 Funktionen zur Verfügung stellt.

Format:

BATCHMAN [Befehl] [Parameter] /R

Hinweise:

Für die Funktion von BATCHMAN ist die DOS ERRORLEVEL-Variable wichtig. Über diese Variable teilt BATCHMAN der Batchdatei das Ergebnis des Befehls mit. Im Folgenden kürze ich ERRORLEVEL mit EI ab. In der Beschreibung der Befehle sind die entsprechende Rückgabewerte von EI in den geschweiften Klammern angegeben. Der Parameter /R bewirkt, daß der Wert von EI auf dem Bildschirm angezeigt wird (zum Testen der Batchdateien).

Die Befehle:

ANSI {EI=0 wenn ANSI-Treiber geladen, sonst EI=1}
Stellt fest, ob ein ANSI-Bildschirmtreiber geladen ist.

BEEP [m,n[;m,n[...]]] {EI=0}
BEEP gibt beliebige Töne auf dem Lautsprecher aus. m ist die Frequenz in Hz, n die Länge in 1/18 sek.
BATCHMAN BEEP 1046,9
gibt eine C-Note 1/2 Sekunde aus. Man kann auch ganze Melodien spielen:
BATCHMAN BEEP
392,3;523,3;659,3;784,3;784,3;10,3;659,3;784,12
Bei Frequenzen <19 Hz wird kein Ton ausgegeben sondern nur die Zeit gewartet.

BREAK {EI=0 wenn BREAK=OFF, sonst EI=1}
Ermittelt den Stand der DOS-Environmentvariable BREAK.

CANCOPY dateispez. [d:]
{EI=0 wenn Platz ausreicht, sonst EI=1}
Prüft, ob genug Platz auf dem angegebenen/aktuellen Laufwerk zum Kopieren der Dateien ist.
BATCHMAN CANCOPY C:\TEMP*.* A:
Prüft, ob genügend Platz auf Laufwerk A: ist. Wenn ja, dann EI=0; wenn nein, dann EI=1.

CECHO [C] [nn,]String {EI=0}
CECHO gibt den String an der aktuellen Cursorposition aus. Mit nn kann eine Farbe gewählt werden (Farbnummer siehe CLS). Bei Angabe von C wird keine Zeilenschaltung ausgegeben.

```
ECHO OFF
BATCHMAN CECHO C CEh,Hello
BATCHMAN CECHO 4Eh, World !
```

Gibt "Hello World !" aus, wobei "Hello" rot auf gelb blinkend und " World !" rot auf gelb dargestellt sind.

CLS [nn] {El=0}
CLS löscht den Bildschirm und zwar auch in den EGA 43 Zeilen, bzw. VGA 50 Zeilen Modi. Mit nn kann eine Farbe angegeben werden, wobei nn = Vordergrundfarbe + (Hintergrundfarbe * 16). Man kann nn auch Hexadezimal angeben:
Dezimal: BATCHMAN CLS 113
Hexadez.: BATCHMAN CLS 71h
Vordergrund=1; Hintergrund = 7

COMPARE String1 String2 {El=1 wenn gleich, sonst El=0}
Vergleicht String1 mit String2 ohne Beachtung der Groß-/-Kleinschreibung, im Gegensatz zu dem DOS-Befehl IF String1==String2, der die Groß-/Kleinschreibung beachtet.

COLDBOOT / WARMBOOT
WARMBOOT bewirkt dasselbe, wie ein Druck auf die Tasten <Ctrl-Alt-Del>, bzw. deutsch <Strg-Alt-Lösch>. COLDBOOT bewirkt dasselbe, wie ein Druck auf den Resetknopf, bzw. Ein- und Aus- schalten des Rechners. Da beim WARMBOOT kein Speichertest durch- geführt wird ist er schneller als der COLDBOOT.

CPU20 {El=CPU-Typ}.
Ermittelt den CPU-Typ:
1=8088/8086; 2=80186; 3=80286; 4=80386/80486

CURSORTYPE [m,n] {El=0}
Setzt den Cursortyp. Ohne Parameter wird der Standardcursor eingestellt (Unterstrich). Ansonsten wird ein Cursor mit m/n als Start-/Stoplinie eingestellt.

DIREXIST Verzeichnis {El=1 wenn existent, sonst El=0}

DRIVEEXIST d:
Prüft, ob das angegebene Verzeichnis/Laufwerk existiert.

DISPLAY {El=Bildschirmtyp}
Ermittelt den Bildschirmtyp:
1=MDA/Hercules; 2=CGA; 4=EGA color; 5=EGA mono; 6=PGS; 7=VGA mono; 8=VGA color; 11=MCGA mono; 12 = MCGA color

DOSVER {El=(major*32)+minor}
Ermittelt die DOS-Version. Bei DOS 3.30 ergibt sich major=3 und minor=30, also El=(3*32)+ 30 = 96+30 = 126

E43V50
Schaltet bei EGA in den 43- und bei VGA in den 50-Zeilen Modus.

GETKEY ["String" n] {El=Scan Code/Position in der Liste}
Wird keine Liste angegeben, liefert GETKEY den SCAN-Code der gedrückten Taste. Mit SHIFT ALT, bzw. SHIFT CTRL kann man überprüfen, ob ALT oder CTRL gedrückt waren.

```
ECHO OFF
:GETKEY
BATCHMAN GETKEY
IF NOT ERRORLEVEL 16 GOTO GETKEY
BATCHMAN SHIFT ALT
IF NOT ERRORLEVEL 1 GOTO GETKEY
ECHO Alt-Q wurde gedrückt
```

Wenn eine Liste angegeben wird gibt ERRORLEVEL die Position der Taste in der Liste an.
BATCHMAN GETKEY "yn" F1
Gibt El=1 für <Y>, El=2 für <N>, El=3 für <F1> oder El=255 für <Ctrl-Break>.

ISVOL [d:]Volume {El=1 wenn Volume identisch, sonst El=0}
Prüft, ob der angegebene Volume Label gleich dem Volume Label des angegebenen/aktuellen Laufwerkes ist.

MAINMEN n|R
{El=0 wenn genügend freier Speicher, sonst El=1}

EXPMEM n|R

EXTMEM n|R
Prüft, ob n Kilobytes freier Haupt-/EMS-/Erweiterungsspeicher vorhanden sind. Wird R statt n angegeben, zeigt BATCHMAN die freien Kilobytes auf dem Bildschirm an.

NUMLOCK [ON|OFF] {El=0}

SCROLLOCK [ON|OFF]

CAPSLOCK [ON|OFF]
Schaltet Num-/Scroll-/Capslock um, bzw. bei Angabe von ON oder OFF an oder aus.

PRTSC
Druckt den Bildschirminhalt aus (wie die PrtSc-Taste, bzw. deutsch die Druck-Taste).

PUSHPATH {El=0 wenn erfolgreich, sonst El=1}

POPPATH
PUSHPATH merkt sich den aktuellen Pfad, POPPATH ruft gemerkte Pfade wieder ab.

```
BATCHMAN PUSHPATH     | merkt sich den akt. Pfad
D:                    | auf Laufwerk D: wechseln
BATCHMAN PUSHPATH     | den Pfad auf D: merken
CD \WORD5             | nach \WORD5 wechseln
WORD                  | WORD aufrufen
BATCHMAN POPPATH      | Pfad auf D: abrufen
BATCHMAN POPPATH      | Pfad abrufen
```

QFORMAT [d:] [N] {El=0 wenn erfolgreich, sonst El=1}
Löscht eine komplette Diskette. Es werden als Laufwerke nur A: und B: akzeptiert, um ein versehentliches Formatieren der Festplatte zu verhindern. Wird N angegeben löscht BATCHMAN ohne Nachfrage ! Bei Fehler während des Löschen wird El=1 gesetzt.

RENDIR Alt Neu {El=0 wenn erfolgreich, sonst El=1}
Benennt ein Verzeichnis von Alt in Neu um (ab DOS 3.0).

SETLOOP [n] {El=0} DECLOOP {El=Zähler}
SETLOOP setzt einen Schleifenzähler auf n. Mit DECLOOP wird dieser Zähler um 1 verringert und der neue Zähler-

CCD e.V. PRUNE

Funktion:

PRUNE ist ein Utitilty zum Manipulieren von ganzen Verzeichnisästen. Mit PRUNE kann ein Verzeichnis mit allen seinen Unterverzeichnissen, also ein Verzeichnisast, gelöscht, kopiert oder verschoben werden.
Außerdem kann man den durch einen Verzeichnisast belegte Platz auf der Platte ermitteln und Verzeichnisse umbenennen. Die Programme DIRMATCH und DR können direkt von PRUNE aus aufgerufen werden.

Format:

PRUNE [d:] [d:]

Hinweise:

Wird PRUNE ohne Parameter aufgerufen, startet es mit dem aktuellen Laufwerk in beiden Fenstern. Mit den beiden Parametern lassen sich die Laufwerke für die beiden Fenster bestimmen.
Mit der Taste <F7> kann auch nach dem Aufruf ein anderes Laufwerk in das aktuelle Fenster geladen werden.
Mit den Tasten <Rechts>, <Links> oder der <TAB>-Taste kann zwischen den zwei Fenstern hin- und hergeschaltet werden.
Mit den <Auf>, <Ab>, <Bild auf>, <Bild ab>, <Pos 1>, <Ende> Tasten kann man in den Verzeichnisbäumen blättern.
!!! Die Befehle <F1>, <F2>, <F4> und <F5> beziehen sich auf den kompletten Ast, d.h. das markierte Verzeichnis mit allen seinen Unterverzeichnissen !!!
Mit <F1> kann man den markierten Ast an eine andere Stelle des aktiven Verzeichnisbaumes oder in den anderen Verzeichnisbaum kopieren.
<F2> löscht den markierten Verzeichnisast (Vorsicht !)
<F3> benennt das markierte Verzeichnis um.
<F4> verschiebt den markierten Verzeichnisast. Das Ziel kann auch auf einem anderen Laufwerk liegen.
<F5> ermittelt den vom markierten Ast auf der Platte belegten Platz.
<F6> wie <F5>, aber nur für das markierte Verzeichnis.
<F7> lädt eine anderes Laufwerk in das aktive Fenster.
<F8> startet DR.
<F9> startet DIRMATCH.
Für <F8> und <F9> müssen DR.COM und DIRMATCH.COM im Pfad sein.

Quelle:

PC-Magazine, New York, Volume 9, No. 12

CCD e.V. CHKFRAG

Funktion:

CHKFRAG ermittelt die Zersplitterung der Dateien auf der Festplatte. Die Zersplitterunggrad wird auf dem Bildschirm ausgegeben. Er kann auch über die ERRORLEVEL Variable von einer Batchdatei abgefragt werden. Auf diese Weise kann die Sortierung der Platte automatisiert werden.

Format:

```
CHKFRAG [d:] [/% | /N | /E | /F] [/L] [/4]
```

Hinweise:

Das Betriebssystem unterteilt die Festplatte in sogenannte Cluster. Diese Cluster haben typischerweise eine Größe zwischen 2 und 8 kByte. In der File Allocation Table (Dateizuordnungstabelle) hält DOS fest, welche Dateien welche Cluster belegen.
Eine Datei muß nicht unbedingt aufeinanderfolgende Cluster belegen. Sie kann auch quer über die Festplatte verstreut sein. Solche Dateien werden "fragmented files" genannt.
Der Zugriff auf "fragmented files" geschieht normalerweise langsamer als bei zusammenhängeden Dateien, da der Schreib-/Lesekopf viele Spurwechsel zurücklegen muß.
Außerdem können zwischen sich zwischen den Dateien noch unbelegte Cluster befinden, sogenannte "free space areas".
"fragmented files" und "free space areas" verlangsamen den Zugriff auf die Festplatte.

Wird CHKFRAG ohne Parameter aufgerufen, überprüft es das aktuelle Laufwerk und gibt eine Statistik mit der Zahl der Dateien und Verzeichnisse, wieviele und wieviel Prozent davon zersplittert sind, wieviele nicht zusammenhängende Bereiche existieren, wieviele freie Bereiche zwischen den Dateien existieren.
Der Parameter d: bewirkt, daß CHKFRAG das Laufwerk d überprüft.
CHKFRAG setzt beim Beenden die ERRORLEVEL-Variable von DOS auf einen bestimmten Wert. Das kann entweder der Prozentsatz der zersplitterten Dateien, die Zahl der zerplitterten Dateien, die Zahl der nicht zusammenhängenden Bereiche oder die Zahl der freien Bereiche sein.
Der Parameter /% setzt ERRORLEVEL auf den Prozentsatz der zersplitterten Dateien (/% ist Voreinstellung).
Der Parameter /N setzt ERRORLEVEL auf die Zahl der zersplitterten Dateien.

Der Parameter /E setzt ERRORLEVEL auf die Zahl der nicht zussammenhängenden Bereiche.

Der Parameter /F setzt ERRORLEVEL auf die Zahl der freien Bereiche.
Der Parameter /L bewirkt, daß die Namen der zersplitterten Dateien aufgelistet werden.
Der Parameter /4 wird benötigt, wenn die Platte mit DOS 4.0 formatiert wurde.

Quelle:

PC-Magazine, New York, Volume 8, Heft 14
Version 1.1: Verbessert von Michael Holmes

Funktion:

Ermittelt die Geschwindigkeit der im System befindlichen Festplatten.

Format:

[*d:*][*pfad*]CORETEST [*/B:xx*] [*/C:xxx*] [*/D:n*[*="xxx"*]] [*/P:xx*] [*/T:xxx*]
[*d:*][*pfad*]CORETEST [*?*]
[*d:*][*pfad*]CORETEST [*HELP*]

Typ: Intern Extern

***

Hinweise:

CORETEST ermittelt die Leistung der Festplatten mit drei Tests: Datenübertragungsrate, Spur-zu-Spur Wechselzeit und mittlere Zugriffszeit. CORETEST zerstört keinerlei Daten auf der Festplatte und kann sooft aufgerufen werden, wie man will.

Die Datentransferrate wird durch wiederholtes Lesen desselben Zylinders ermittelt. Sie hängt von der Geschwindigkeit des Rechners und des Festplattenkontrollers, sowie des verwendeten Interleave-Faktors ab.

Die Puffergrößee wird automatisch auf die Größe eines Zylinders gesetzt. Übersteigt die Größe 64 Kb, wird sie auf 64 Kb gesetzt. Mit dem Parameter /B:xx kann man die Puffergröße auch manuell setzen, wobei xx die Puffergröße in Kb (maximal 64 Kb) darstellt.

Der Übertragungstest wird normalerweise über 4 Sekunden durchgeführt. Mit /T:xxx kann man die Zeit auf xxx Sekunden umstellen.

Die mittlere Zugriffszeit wird durch 200 Spurwechsel über 1/3 der Spuren gemessen. Um die mittlere Zugriffszeiten in einem begrenzten Spurenbereich zu ermitteln, kann man mit dem Parameter /C:xxx die Spurenzahl auf xxx Spuren umstellen.

Mit dem Parameter /D:n[="xxx"] kann man ein bestimmtes Laufwerk testen. n ist die physikalische Laufwerksnummer (0 oder 1). Der optionale Parameter = "xxx" definiert einen Namen der Platte für die Geschwindigkeitstabelle. Der Parameter /D kann zweimal in der Kommandozeile erscheinen.

Mit dem Parameter /P:xx kann man eine Pause von xx Sekunden nach dem Test einstellen.

Wird der Parameter ? oder HELP angegeben, zeigt CORETEST die möglichen Parameter an.

CORETEST erkennt automatisch, ob ein Cache vorhanden ist und meldet dies auf dem Bildschirm.

Aus den Geschwindigkeitsdaten errechnet CORETEST einen Leistungsfaktor. Faktor 1 hat dabei ein IBM-XT mit 10 Mb Festplatte.

CORETEST zeigt am Ende noch eine Tabelle an, in der man seine Werte mit denen von anderen Rechnern, die CORE getestet hat vergleichen kann.

oder einen String n mal.

Liste
Num → Liste
oder String
Num → String

Bsp.: { 1 2 3 }
3
→
{ 1 2 3 1 2 3 1 2 3 }

oder "Hallo "
3
→
"Hallo Hallo Hallo "

CONVERSE
kehrt die Reihenfolge der ersten n Objekte auf dem Stack um.

.
. .
. .
n → .

Bsp.: 5
4
3
2
1
3
→
5
4
1
2
3

REVERSE
kehrt die Reihenfolge der Elemente einer Liste um.

Liste → Liste

Bsp.: { 1 2 3 4 5 }
→
{ 5 4 3 2 1 }

SEQUENCE
nimmt drei Zahlen vom Stack und erzeugt eine Folge von Zahlen.

Num
Num
Num → Liste

Bsp.: 0
3
10
→
{ 0 3 6 9 }

INDEX erzeugt eine Indexmenge (-Liste) von einer Liste oder zwei Zahlen, die als Intervallgrenzen gewertet werden.

Liste → Liste
oder Num
Num → Liste

Bsp.: { "Hallo" 1 2 3 { 987 } }
→
{ 1 2 3 4 5 }
oder 4
11
→
{ 4 5 6 7 8 9 10 11 }

INTERSECT
erzeugt die Schnittmenge aus zwei Listen. Genaugenommen werden aus der ersten Liste diejenigen Elemente übernommen, die auch in der zweiten gefunden werden. Auch doppelt vorkommendede, jedoch werden nicht diejenigen doppelt übernommen, die in der zweiten Liste doppelt vorkommen. Eine echte SchnittMENGE ergibt sich also nur bei echten Mengen, die keine Doppeleinträge enthalten (siehe MKSET).

Liste1
Liste2 → Liste

Bsp.: { 1 1 2 3 4 }
{ 1 2 2 5 6 }
→
{ 1 1 2 }

SUBTRACT
subtrahiert zwei Listen, das bedeutet: das jeweils erste Vorkommnis eines jeden Elementes der zweiten Liste in der ersten wird aus der ersten entfernt.

Liste1
Liste2 → Liste

Bsp.: { 1 2 4 4 2 1 6 5 3 2 }
{ 1 3 2 4 2 3 1 5 3 2 }
→
{ 4 2 6 5 2 }

MKSET (make set) macht aus einer Liste eine (endliche) Menge im mathematischen Sinne. D.h.: es werden Doppeleinträge entfernt.

Liste → Liste

Bsp.: { 1 2 3 1 2 4 2 1 3 4 6 3 2 1 }
→
{ 4 2 1 3 4 6 3 2 1 }

TRANSPOSE
transponiert eine Liste von Listen. Das ist nicht nur bei Matritzen sinnvoll, auch außerhalb der Mathematik gibt es für diese Funktion Anwendungen (siehe Listing von FOLDRN und FOLDLN).

Liste → Liste

Bsp.: {{ 1 2 }{ 3 4 }{ 5 6 }}
→
{{ 1 3 5 }{ 2 4 6 }}

SORT sortiert eine Liste mit « < » in aufsteigender Reihenfolge. Handelt es sich bei den Listenelementen wiederum um Listen, so wird nach deren ersten Element sortiert. SORT ruft SORTWITH auf.

Liste → Liste

Bsp.: { 2 3 1 7 4 9 8 5 6 }
→
{ 1 2 3 4 5 6 7 8 9 }

SORTWITH
erwartet auf dem Stack die zu sortierende Liste und den Vergleichsoperator, mit dessen Hilfe sortiert werden soll. Dadurch kann für alle erdenklichen zu sortierenden Daten eine geeignete Testroutine geschrieben werden, ohne daß man sich über den Sortieralgorithmus noch Gedanken machen muß. Es wird ein Binary Insertion Sort mit vorherigem Test, ob ein Einfügen überhaupt nötig ist, gemacht. Dadurch werden nahezu sortierte Listen sehr schnell sortiert. Da der Algorithmus von hinten nach vorne sortiert, werden neue Elemente in eine bereits sortierte Liste am schnellsten einsortiert, indem man sie vor dem Sortieren vorne an die Liste anfügt.

Liste
Prädikat → Liste

Bsp.: {{ 4 2 3 }{ 3 1 65 }{ 0 5 3 5 3 }}
{ 2 GET SWAP 2 GET > }
→
{{ 3 1 65 }{ 4 2 3 }{ 0 5 3 5 3 }}
(die Listen sind nun nach ihrem zweiten Element sortiert)

A\-L verwandelt ein ein- oder zweidimensionales Array in eine lineare Liste.

Array → Liste

Bsp.: [[1 2][3 4][5 6]]
→
{ 1 2 3 4 5 6 }

A\-L2 verwandelt eineindimensionales Array in eine lineare Liste (wie A\-L), oder ein zweidimensionales Array in eine Liste von Listen.

Array → Liste

Bsp.: [[1 2][3 4][5 6]]
→
{{ 1 2 }{ 3 4 }{ 5 6 }}

L\-A verwandelt eine Liste von numerischen Werten in ein eindimensionales Array, oder eine Liste von Arrays oder Listen, die numerische Daten enthalten, in ein zweidimensionales Array. Achtung: die Unterlisten oder Arrays müssen alle gleiche Größe haben!

Liste → Array

Bsp.: {{ 1 2 }{ 3 4 }{ 5 6 }}
→
[[1 2][3 4][5 6]]

MAP

Checksum: # 3CF9h Size: 100 Bytes

```
« SWAP LIST→
  IF DUP
  THEN 2 + DUP ROLL 3 3 PICK
    START OVER ROLL OVER EVAL ROT ROT
    NEXT DROP 2 - →LIST
  ELSE DROP2 { }
  END
»
```

MAP2

Checksum: # 2062h Size: 198.5 Bytes

```
« ROT LIST→ 3 + DUP ROLL LIST→ 6 + DUP 4 - ROLL OVER +
  4 - DUP 2 - ROLL SWAP ROT DUP2 6 - SWAP 5 - OVER -
  MIN LASTARG MAX OVER - 5 ROLLD DUP 5 ROLLD
  IF
  THEN LASTARG 1 SWAP
    START OVER ROLL OVER ROLL 5 PICK EVAL 5 ROLLD SWAP
      1 - SWAP
    NEXT
  END DROP DROP2 →LIST OVER 2 + ROLLD DROPN
»
```

FILTER

Checksum: # 1E07h Size: 153 Bytes

```
« SWAP LIST→
  IF DUP
  THEN 3 + 0 SWAP DUP ROLL 4 3 PICK
    START OVER ROLL OVER OVER SWAP
      IF EVAL
      THEN 4 ROLLD ROT 1 + ROT ROT
      ELSE DROP SWAP 1 - SWAP
      END
    NEXT DROP2 →LIST
  ELSE DROP2 { }
  END
»
```

SPLIT

Checksum: # 1FFFh Size: 194.5 Bytes

```
« SWAP LIST→
  IF DUP
  THEN 3 + 4 SWAP DUP ROLL 4 3 PICK
    START OVER ROLL OVER OVER SWAP
      IF EVAL
      THEN 4 ROLLD ROT 1 + ROT ROT
      ELSE 4 PICK ROLLD
      END
    NEXT DROP OVER 2 - ROLLD LASTARG DUP ROLLD 4 -
      →LIST ROT ROT - ROLLD LASTARG 1 - →LIST
  ELSE DROP2 { } { }
  END
»
```

FOLDL1

Checksum: # 57F0h Size: 128.5 Bytes

```
« → f
  « LIST→
    IF DUP 1 >
    THEN ROLL LASTARG 2
      FOR i i ROLL f EVAL -1
      STEP
    ELSE
      IF NOT
      THEN "Empty List" DOERR
      END
    END
  »
»
```

FOLDR1

Checksum: # 3074h Size: 112 Bytes

```
« → f
  « LIST→
    IF DUP 1
    THEN 2 SWAP
      START f EVAL
      NEXT
    ELSE
      IF NOT
      THEN "Empty List" DOERR
      END
    END
  »
»
```

FOLDL

Checksum: # AAD8h Size: 90 Bytes

```
« → f
  « SWAP LIST→
    IF
    THEN LASTARG 1 + ROLL LASTARG 2
      FOR i i ROLL f EVAL -1
      STEP
    END
  »
»
```

FOLDR

Checksum: # 6FF1h Size: 78.5 Bytes

```
« → f
  « SWAP LIST→
    IF
    THEN LASTARG 1 + ROLL LASTARG 2 SWAP
      START f EVAL
      NEXT
    END
  »
»
```

FOLDLN

Checksum: # 45B5h Size: 64 Bytes

```
« SWAP TRANSPOSE SWAP 'FOLDL1' 2 →LIST MAP
»
```

FOLDRN

Checksum: # 2B5Fh Size: 64 Bytes

```
« SWAP TRANSPOSE SWAP 'FOLDR1' 2 →LIST MAP
»
```

SCAN

Checksum: # B2B9h Size: 107.5 Bytes

```
« → f
  « LIST→
    IF DUP
    THEN 1 + ROLL LASTARG 2 + 4 OVER
      START DUP2 ROLL f EVAL SWAP
      NEXT 2 -
    ELSE DROP 1
    END →LIST
  »
»
```

SCAN1

Checksum: # 6B5Dh Size: 96 Bytes

```
« → f
  « LIST→
    IF DUP 1
    THEN ROLL LASTARG 2 + 4 OVER
      START DUP2 ROLL f EVAL SWAP
      NEXT 2 -
    END →LIST
  »
»
```

LINES

Checksum: # 55ABh Size: 187.5 Bytes

In diesem Listing steht der Punkt (•) für Carriage Return (Zeilenschaltung)

```
«
  IF DUP DUP SIZE DUP SUB "•" ≠
  THEN "•" +
  END 0 1000000000
  FOR i
    IF DUP "•" POS
    THEN LASTARG 1 - 1 SWAP SUB LASTARG SWAP DROP 2 +
      100000 SUB
    ELSE DROP i →LIST 1000000000 'i' STO
```

```
    END
  NEXT
»
```

LAY

Checksum: # A41Ah Size: 67.5 Bytes

In diesem Listing steht der Punkt (•) für Carriage Return (Zeilenschaltung)

```
« "" { →STR "•" SWAP + + } FOLDL 2 1000000 SUB
»
```

CONCAT

Checksum: # 1987h Size: 111.5 Bytes

```
« { } { SWAP
  IF DUP TYPE 5 ≠
  THEN 1 ROT
  ELSE LIST→ DUP 2 + ROLL
  END LIST→ DUP 2 + ROLL + →LIST } FOLDR
»
```

CONCAT2

Checksum: # 68D0h Size: 124 Bytes

```
« SWAP
  IF DUP TYPE 5 ≠
  THEN 1 ROT
  ELSE LIST→ DUP 2 + ROLL
  END
  IF DUP TYPE 5 ≠
  THEN 1 ROT
  ELSE LIST→ DUP 2 + ROLL
  END + →LIST
»
```

PREFIX

Checksum: # 4A24h Size: 30.5 Bytes

```
« LIST→ 1 + →LIST
»
```

SUFFIX

Checksum: # 2E5Dh Size: 38 Bytes

```
« SWAP LIST→ 1 + ROLL LASTARG →LIST
»
```

HD

Checksum: # 74CBh Size: 29 Bytes

```
« 1 1 SUB 1 GET
»
```

TL

Checksum: # B6C2h Size: 32 Bytes

```
« 2 100000 SUB
»
```

INIT

Checksum: # C31Ah Size: 33.5 Bytes

```
« 1 OVER SIZE 1 - SUB
»
```

LST

Checksum: # 12D1h Size: 32.5 Bytes

```
« DUP SIZE DUP SUB 1 GET
»
```

DROPWHILE

Checksum: # F6DCh Size: 147.5 Bytes

```
« → t
  « LIST→ DUP
    IF
    THEN 0 SWAP 1 + 2
      FOR i i ROLL
        IF DUP t EVAL
        THEN DROP
        ELSE i ROLLD DROP i 1 - 0 'i' STO
        END -1
      STEP
    END →LIST
  »
»
```

TAKEWHILE

Checksum: # 6BEFh Size: 171.5 Bytes

```
« OVER SIZE → t s
  «
    IF s
    THEN LIST→ 1 + 0 SWAP 2
      FOR i
        IF i PICK t EVAL NOT
        THEN DROP i 1 - 0 'i' STO
        END -1
      STEP DUP DUP 2 + ROLLD DROPN s SWAP - →LIST
    END
  »
»
```

RPT

Checksum: # F184h Size: 65 Bytes

```
« OVER SIZE OVER * ROT ROT 1 SWAP LOG2 CEIL
  START DUP +
  NEXT 1 ROT SUB
»
```

CONVERSE

Checksum: # C8F5h Size: 147 Bytes

```
« → n
  «
    IF DEPTH n
      IFERR IP
      THEN SWAP DROP ERRN DOERR
      END <
     THEN n 515 DOERR
    END
    IF n IP 1 >
    THEN 2 n
      FOR i i ROLL
      NEXT
    END
  »
»
```

REVERSE

Checksum: # 3AC7h Size: 87 Bytes

```
« LIST→ DUP → s
  «
    IF DUP 1 >
    THEN 2 SWAP
      FOR i i ROLL
      NEXT s
    END →LIST
  »
»
```

SEQUENCE

Checksum: # BC8Ah Size: 290.5 Bytes

```
« → a s b
  « s a -
    IF
    THEN LASTARG 's' STO
    ELSE
      IF a b ==
      THEN 1 's' STO
      ELSE a s b # 203h DOERR
      END
    END    IF b a - s * 0 <
    THEN a DUP s + b # 203h DOERR
    END a b
    FOR i i s
    STEP b a - s / IP 1 + →LIST
  »
»
```

INDEX

Checksum: # E49Ch Size: 138.5 Bytes

```
«
  IF DUP TYPE
  THEN 1 SWAP SIZE
  END DUP2 SWAP - DUP ABS 1 + SWAP SIGN → n s
  «
    IF s
    THEN
      FOR i i s
      STEP n
    ELSE DROP 1
    END →LIST
  »
»
```

INTERSECT

Checksum: # 8C1h Size: 52.5 Bytes

```
« { SWAP POS } PREFIX FILTER
»
```

SUBTRACT

Checksum: # FB4Dh Size: 117.5 Bytes

```
« LIST→
  IF
  THEN LASTARG 1 + ROLL LASTARG 2 SWAP
    START
      IF DUP ROT POS
      THEN LASTARG SWAP LIST→ DUP 2 + ROLL LASTARG SWAP
        - ROLL DROP 1 - →LIST
      END
    NEXT
  END
»
```

MKSET

Checksum: # 9786h Size: 99.5 Bytes

```
« LIST→
  IF
  THEN LASTARG { } 1 ROT
    START
      IF SWAP POS LASTARG ROT
      THEN DROP
      ELSE SWAP LIST→ 1 + →LIST
      END
    NEXT
  ELSE { }
  END
»
```

TRANSPOSE

Checksum: # A3B8h Size: 276.5 Bytes

```
« DUP MAXR →NUM { SIZE MIN } FOLDL
  IF
  THEN LASTARG SWAP LIST→ DUP 2 + ROLL → r c
    « 0 r 1 -
      FOR i r i - i c * + ROLL LIST→ c - DROPN
      NEXT 0 c 1 -
      FOR i 0 r 1 -
        FOR j c i - r j - * i j + + ROLL
        NEXT r →LIST
      NEXT c →LIST
    »
  ELSE DROP { { } }
  END
»
```

SORT

Checksum: # 4447h Size: 183.5 Bytes

```
« DUP 1 GET DUP
  IFERR <
  THEN DROP 1
    IFERR GET
    THEN DROP2 # 202h DOERR
    ELSE DUP
      IFERR <
      THEN DROP2 # 202h DOERR
      ELSE DROP { 1 GET SWAP 1 GET > }
      END
    END
  ELSE DROP {  }
  END SORTWITH
»
```

SORTWITH

Checksum: # 6E5Bh Size: 304.5 Bytes

```
« 0 → t s
  «
    IF DUP TYPE 5 SAME
    THEN LIST→ DUP
    ELSE ARRY→ DUP LIST→
      IF 2 SAME
      THEN *
      END SWAP
    END 's' STO 2 SWAP
    FOR i
      IF i PICK i PICK SWAP t EVAL
      THEN i ROLL i 4 + 4
        WHILE DUP2 + 2 / CEIL DUP 4 PICK ≠
        REPEAT DUP PICK 5 PICK
          IF t EVAL
          THEN SWAP ROT
          ELSE SWAP
          END DROP
        END 4 - ROT ROT DROP2 ROLLD
      END
    NEXT s
    IF DUP TYPE 5 SAME
    THEN →ARRY
    ELSE →LIST
    END
  »
»
```

A→L

Checksum: # 22FCh Size: 40 Bytes

```
« ARRY→ LIST→
  IF 2 ==
  THEN *
  END →LIST
»
```

A→L2

Checksum: # 55CDh Size: 116 Bytes

```
« ARRY→ LIST→
  IF 2 ==
  THEN → r c
    « 1 r
      FOR i c →LIST r i - c * i + ROLLD
      NEXT r
    »
  END →LIST
»
```

L→A

Checksum: # 8793h Size: 255.5 Bytes

```
«
  IF { 0 1 } OVER 1 GET TYPE POS
  THEN LIST→
  ELSE LIST→
    IF OVER TYPE 5 ≠
    THEN OVER SIZE LIST→
      IF 2 ==
      THEN # 202h DOERR
      END SWAP 2 + 3 OVER
      START DUP ROLL ARRY→ LIST→ DROP 2 + ROLL LASTARG
        ROLL LASTARG + 3 -
      NEXT
    ELSE OVER SIZE SWAP 2 + 3 OVER
      START DUP ROLL LIST→ 2 + ROLL LASTARG ROLL
        LASTARG + 3 -
      NEXT
    END 2 - OVER / SWAP 2 →LIST
  END →ARRY
»
```

LOG2

Checksum: # 438Dh Size: 45 Bytes

```
« → x 'LOG(x)/LOG(2)'
»
```

Matthias Rabe
Teichsheide 13
4800 Bielefeld

Pas de deux Teil II

Programme für den HP-48SX mit Tips für den HP-28S

von Ralf Pfeiffer

Wie bei jedem schlechten Film gibt es auch von dem in 3/90 erschienenen Artikel eine Fortsetzung, und es beginnt als Lehrstück zum Thema Rekursion. Dieses gilt in gleicher Weise für 28S wie 48SX, auch wenn die Programme nur als 48SX-Listings beiliegen.

Als erstes Beispiel soll der euklidische Algorithmus zur Berechnung des größten gemeinsamen Teilers (ggT) programmiert werden. Die Rekursionsvorschrift lautet dabei so: "Nimm zwei Zahlen z (wie Zähler) und n (wie Nenner) und berechne den Divisionsrest (z mod n). Ist der Rest der Division nicht Null, dann nimm n und den Rest und berechne deren ggT (=Beginn der Rekursion), andernfalls ist n der ggT."

Bevor wir programmieren, noch eine Konvention: Jedes Programm holt sich z aus Ebene 2 und n aus Ebene 1. Dabei ist es beim Programmstart (dem erstmaligen Aufruf) auch zulässig, daß z kleiner n ist.

Wir probieren zunächst die algebraische Syntax mit dem Programm **GGTA**. Dieses greift sich sofort z und n vom Stack und erzeugt die lokalen Variablen z und n. Der algebraische IFTE(..;..;..) Test besteht aus drei Teilen: Eine algebraisch formulierte Testbedingung die entweder eine 0 oder eine andere beliebige reelle Zahl liefert (muß keine 1 sein, jede Zahl außer Null interpretiert der Rechner als 1). Der zweite Teil im IFTE ist die "ja"-Verzweigung (falls der Test keine 0 geliefert hat), und beinhaltet hier den Einstieg in die Rekursion. Den dritten Teil führt der Rechner aus, falls der Test zu einer Null führte. Die drei Testteile sowie die Argumente im rekursiven Aufruf trennen Strichpunkt und Komma, je nach Stellung von Flag -51 (Dezimalkommaflag), aber keine zusätzlichen Striche (').

Wie vorher schon bemerkt, verlangen alle Test im 28S/48SX entweder nach einer 0 oder nach einer anderen reellen Zahl, also nicht notwendigerweise nach einer Eins. Diese kleine Vereinfachung wendet **GGTB** für die Bedingung im IFTE an, den Test ≠0 und 0 ≠ kann man sich beim 28S immer und beim 48SX dann sparen, wenn man die 0 mit einer reellen Zahl vergleicht. **GGTC** spart durch eine kleine Änderung an der Rekursionsformel noch ein paar Bytes, kürzere Möglichkeiten kenne ich allerdings nicht.

Man kann den ggT auch ohne algebraische Syntax berechnen. **GGT1** greift sich wieder z und n vom Stack, dann kommt nach IF der Test, ob der Rest ungleich Null ist. Falls er nicht Null ist (=THEN), kommt n (Ebene 2) und der Divisionsrest (z mod n, Ebene 1) auf den Stack, andernfalls (=ELSE) ist n der ggT.

Hier kann man jetzt optimieren, GGT2 verzichtet zunächst auf 0 ≠ (Zeile 3, Begründung s.o.). Die Funktion THEN verändert die Stackinhalte wie DROP: Der Inhalt von Ebene 1 verschwindet, wird aber als LASTARG gesichert. Das macht sich GGT2 nutzbar, denn Zeile 3 berechnet nur noch den Divisionsrest, und führt diesen dem THEN (Zeile 4) zu. Nun testet THEN und sichert zugleich den Rest als LASTARG. Falls GGT2 nach THEN fortfährt, holt das Programm zunächst n (Rückrufe verändern LASTARG nicht) und dann den Divisionsrest (natürlich mit der Funktion LASTARG). Macht GGT2 bei ELSE (Zeile 6) weiter, braucht es den Divisionsrest nicht mehr, obwohl er auch hier mit LASTARG verfügbar wäre, denn ELSE ist nur eine Art Label, bewirkt aber sonst nichts.

Bei weiterer Betrachtung fällt uns auf, daß z nur ein einziges Mal aufgerufen wird. **GGT3** verzichtet daher auf die lokale Variable z. Das Programm speichert nur n, so daß in Zeile 2 z noch immer in Ebene 1 des Stacks steht, wenn der IF-Test beginnt, genau an der Stelle, an die es in GGT2 von der lokalen Variablen z auch gebracht worden wäre. Nach THEN und ELSE holt GGT3 jeweils n zurück, aber das kann man schon für beide Programmteile vor das IF ziehen.

GGT4 holt daher in Zeile 2 n und vertauscht es mit z, da der Test dieses z in Ebene 1 erwartet. Nach THEN steht n bereits in Ebene 1, es braucht daher nicht mehr geholt werden. Zwischen ELSE und END steht jetzt überhaupt kein Befehl mehr, deshalb kann man sich das ELSE auch noch sparen. Der letzte Schritt zur Optimierung eleminiert die lokale Variable n: Wenn GGT4 beim MOD-Befehl in Zeile 3 ankommt braucht es folgende Stackordnung: Ebene 1 und 3 enthalten n, Ebene 2 enthält z. Das geht aber mit einfachen Stackoperationen, z.B. mit SWAP OVER, so daß **GGT5** die wohl kürzeste und schnellste rekursive Routine für den HP-28S darstellt.

Für den HP-48SX gibt es noch eine Möglichkeit, die man allerdings eher als Verstümmelung denn als Verbesserung charakterisiert, weshalb ich mich auch von meinem Programm nachdrücklich distanziere. **GGT6** verzichtet nämlich auf die IF...THEN-Struktur und verwendet stattdessen IFT. Wenn Ebene 2 eine 0 enthält, bewirkt IFT ein DROP, während jede andere reelle Zahl zu einem EVAL führt. Außerdem arbeitet ein EVAL im 48SX eine Liste wie ein Programm ab, nur daß eine Liste weniger Bytes braucht. In diesem Falle hat das ganze optimieren nicht viel gebracht, denn das schnellste und kürzeste Programm zur Berechnung des ggT ist nicht rekursiv, es ist **GGT**. Doch es gibt Probleme, die mit Rekursion kürzer gelöst werden können, wer's nicht glaubt kann sich gerne an SORT versuchen!

Als Abschluß des ggT-Problems jetzt noch ein Hammer: Neben Rekursion und der iterativen Lösung jetzt eine, die ein schlafendes Problem auf den Stack legt, nämlich **GGTS**. Diese Technik ist hier zwar nicht sehr effektiv, aber ich habe mit dieser bereits das Programm DKAL (abgedruckt in PRISMA 90.3.26) erstellt und gegenüber den rekursiven Vorversionen ca. 80 Bytes einsparen können. Das in der gleichen Ausgabe abgedruckte PRDIR konnte ich inzwischen ebenfalls mit dieser Methode umrüsten und 36 Bytes abspecken.

Doch nun zur "Technologie" von GGTS: Zeilen 2-7 enthalten das schlafende Programm, welches in Zeile 8 in Ebene 3 geschrieben wird. Die Initialzündung erfolgt in Zeile 9 mit EVAL, dann entscheidet das kleine Programm selbst, ob es sich kopiert und startet (Zeilen 4-5) oder ob es sich selbst löscht und den Programmlauf abbricht.

Dahinter steht folgender Gedanke: 28S und 48SX fertigen sich nach dem Programmstart eine Arbeitskopie des Programms an, die sie dann abarbeiten. Löscht sich ein Programm zuerst selbst und erledigt dann wichtige Dinge, so läuft es einmal einwandfrei durch. Beim 41er war ja Schluß, wenn ein Programmm sich selbst mit PCLPS löschte. Als weitere Konsequenz ergibt sich, daß das Editieren von angeHALTenen Programmen zwar das Programm im Speicher verändert, nicht aber die Arbeitskopie des Rechners. Setzt man ein solches Programm nach HALT mit CONT fort, so läuft die ursprüngliche Version weiter. Merke: Wenn der HALT-Indikator leuchtet, dann nach dem Editieren immer KILL!

Die folgenden acht Programme verzichten auf Kniffe zum Kürzen und Beschleunigen. Folgendes mathematische Problem liegt allen vier Formeln zugrunde: Man soll eine Funktion integrieren, aber

1. $$\int x^n e^{cx}\,dx = \begin{cases} n \geq 1 & : \frac{1}{c}\,x^n e^{cx} - \frac{n}{c}\int x^{n-1} e^{cx}\,dx \\ n = 0 & : \frac{1}{c}\cdot e^{c\cdot x} \end{cases}$$

2. $$\int \tan^n cx\,dx = \begin{cases} n \geq 2 & : \frac{1}{c(n-1)}\tan^{n-1} cx - \int \tan^{n-2} cx\,dx \\ n = 1 & : -\frac{1}{c}\cdot \ln\left|\cos(c\cdot x)\right| \\ n = 0 & : x \end{cases}$$

3. $$\int x^n \sin cx\,dx = \begin{cases} n \geq 1 & : \frac{n}{c}\int x^{n-1}\cos cx\,dx - \frac{x^n}{c}\cos cx \\ n = 0 & : -\frac{1}{c}\cdot\cos(c\cdot x) \end{cases}$$

4. $$\int x^n \cos cx\,dx = \begin{cases} n \geq 1 & : \frac{x^n \sin cx}{c} - \frac{n}{c}\int x^{n-1}\sin cx\,dx \\ n = 0 & : \frac{1}{c}\cdot\sin(c\cdot x) \end{cases}$$

```
48SX XEXP
     « → x n c
002    «
         IF n 1 ≥
004      THEN x n ^ c x
     * EXP * c / x n 1 -
006  c XEXP n * c / -
         ELSE c x * EXP
008  c /
         END
010    »
     »  Bytes : 159
012  Checksum : # A1EFh

48SX EXPX
     « → x n c 'IFTE(n≥1
002  ;x^n*EXP(c*x)/c-
     EXPX(x;n-1;c)*n/c;
004  EXP(c*x)/c)'
     »  Bytes : 161,5
006  Checksum : # 31F7h

'X' 3 1  XEXP

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


48SX NTAN
     « → x n a
002    «
         CASE n 0 ==
004        THEN x
           END n 1 ==
006        THEN a x *
     COS ABS LN a / NEG
008        END a x * TAN
     n 1 - ^ n 1 - a * /
010  x n 2 - a NTAN -
         END
012    »
     »  Bytes : 181,5
014  Checksum : # 1247h

48SX TANN
     « → x n c 'IFTE(n==
002  0;x;IFTE(n==1;-LN(
     ABS(COS(c*x)));TAN(
004  c*x)^(n-1)/(c*(n-1)
     )-TANN(x;n-2;c)))'
006  »  Bytes : 184,5
     Checksum : # 119h

RAD  ,4 5 -2  NTAN

1: -,056147709805
```

```
48SX XSIN
     « → x n c
002    «
         IF n 1 ≥
004      THEN n c / x n
     1 - c XCOS * x n ^
006  c x * COS * c / -
         ELSE c x * COS
008  NEG c /
         END
010    »
     »  Bytes : 161,5
012  Checksum : # 606Dh

48SX XCOS
     « → x n c
002    «
         IF n 1 ≥
004      THEN x n ^ c x
     * SIN * c / n c / x
006  n 1 - c XSIN * -
         ELSE c x * SIN
008  c /
         END
010    »
     »  Bytes : 159
012  Checksum : # F3Dh

48SX SINX
     « → x n c 'IFTE(n≥1
002  ;n/c*COSX(x;n-1;c)-
     x^n*COS(c*x)/c;-COS
004  (c*x)/c)'
     »  Bytes : 164
006  Checksum : # 8909h

48SX COSX
     « → x n c 'IFTE(n≥1
002  ;x^n*SIN(c*x)/c-n/c
     *SINX(x;n-1;c);SIN(
004  c*x)/c)'
     »  Bytes : 161,5
006  Checksum : # 938Ah

RAD 'X' 4 'n'  XCOS

'X^4*SIN(n*X)/n-4/n*(3/n
*(X^2*SIN(n*X)/n-2/n*(1/
n*(SIN(n*X)/n)-X*COS(n*X
)/n))-X^3*COS(n*X)/n)'

48SX GGT6
     « SWAP OVER MOD {
002  LASTARG GGT6 } IFT
     »  Bytes : 35
004  Checksum : # 8B87h
```

```
48SX GGTA
     « → z n 'IFTE(z MOD
002  n≠0;GGTA(n;z MOD n)
     ;n)'
004  »  Bytes : 93,5
     Checksum : # B7DDh

48SX GGTB
     « → z n 'IFTE(z MOD
002  n;GGTB(n;z MOD n);n
     )'
004  »  Bytes : 88,5
     Checksum : # EE61h

48SX GGTC
     « → z n 'IFTE(n;
002  GGTC(n;z MOD n);z)'
     »  Bytes : 81,5
004  Checksum : # DE42h

48SX GGT1
     « → z n
002    «
         IF z n MOD 0 ≠
004      THEN n z n MOD
     GGT1
006      ELSE n
         END
008    »
     »  Bytes : 86
010  Checksum : # 26AEh

48SX GGT2
     « → z n
002    «
         IF z n MOD
004      THEN n LASTARG
     GGT2
006      ELSE n
         END
008    »
     »  Bytes : 72
010  Checksum : # BB71h

48SX GGT3
     « → n
002    «
         IF n MOD
004      THEN n LASTARG
     GGT3
006      ELSE n
         END
008    »
     »  Bytes : 63
010  Checksum : # BCC3h

48SX GGT4
     « → n
002    « n SWAP
         IF n MOD
004      THEN LASTARG
     GGT4
006      END
       »
008  »  Bytes : 58,5
     Checksum : # C1F2h

48SX GGT5
     « SWAP OVER
002    IF MOD
         THEN LASTARG GGT5
004      END
       »
006  »  Bytes : 40
     Checksum : # C8E3h

48SX GGT
     «
002    DO SWAP OVER MOD
       UNTIL DUP NOT
004    END DROP
       »
006  »  Bytes : 32,5
     Checksum : # 6E4Dh

48SX GGTS
     «
002    «
         IF DUP
004      THEN SWAP OVER
     MOD 3 PICK EVAL
006      ELSE ROT DROP2
         END
008    » DUP 4 ROLLD
     EVAL
010  »  Bytes : 75
     Checksum : # 2A8h
```

als Lösung gibt es nur eine Rekursionsformel, mit der man das Problem schrittweise vereinfachen kann. Das Integral links taucht in der Lösung rechts mit einem etwas kleineren n wieder auf. Sollte irgendwann n einen speziellen Wert annehmen, so kann das Problem abgeschlossen werden. Dieser spezielle Wert ist in allen Fällen n=0 und in Formel 2 zusätzlich n=1. Dabei muß n eine positive Ganzzahl, c eine beliebige Konstante (reelle Zahl) und x eine reelle Variable sein. Alle Programme haben den Aufbau einer User-Defined-Function, denn der erste Schritt ist dabei immer, alle Argumente vom Stack zu holen und in lokalen Variablen zu speichern, doch dazu später. Alle Programme erwarten in Ebene 3 das, was in den Formeln x heißt, entweder als 'Name' oder reelle Zahl. Ebene 2 muß n als positive Ganzzahl (falls nicht, beginnt Endlosschleife bis Speicherüberlauf, hier erfolgt keine Überprüfung) enthalten und Ebene 1 enthält als Konstante c entweder als 'Name' oder als reelle Zahl.

Nun zur ersten Formel. **XEXP** prüft zunächst ob n größer oder gleich 1 ist, falls ja, erfolgt die Rekursion nach dem THEN, indem man x, n-1 und c in den Stack schreibt und aufruft. Den Spezialfall n=0 behandelt XEXP nach ELSE.

Formel 1 läßt sich auch algebraisch formulieren. Dazu braucht **EXPX** den algebraischen 'IFTE(..;..;..)' Test. Gegenüber XEXP ist EXPX erheblich langsamer und platzfressender. Üblicherweise kann man aber solche algebraischen User-Defined-Functions ableiten. Merkwürdig ist aber, daß bei den hier vorgestellten rekursiv-algebraischen Programmen es auf dem 48SX nicht, und auf dem 28S nur halbherzig geht, was ich für einen echten 48SX-Bug halte.

Während XEXP n nur um 1 pro Selbstlauf reduzierte, verlangt Formel 2, daß man n immer um 2 verkleinert. Deshalb braucht man auch zwei verschiedene einfache Fälle (einen, falls n gerade und einen, falls n ungerade), um mit der Rekursionsformel zu einer Lösung zu gelangen. Dies ist ein idealer Fall für die CASE-Struktur, die in **NTAN** zunächst prüft, ob Spezialfall n=0 vorliegt, dann n=1 überprüft, und dann erst den allgemeinen Fall (er steht bei CASE zwischen den letzten beiden END-Befehlen) bearbeitet.

Für die algebraische Version **TANN** habe ich mir schon etwas mehr Mühe geben müssen, denn ich kenne bisher keinen Weg, CASE in algebraischer Syntax zu formulieren. Aber CASE stellt nur eine vereinfachte IFTE-Struktur dar: Man kann CASE immer durch verschachtelte IF...THEN.IF...THEN.IF....-Strukturen ersetzen, und das gebraucht TANN. Im ersten IFTE stellt es fest, ob n=0 ist, falls nein, kommt der nächste IFTE-Test, der nach n=1 fragt, und sich dann für diesen Spezialfall oder den allgemeinen Fall entscheidet.

Wechselrekursion bedeutet, daß Programm A auf Programm B zugreift, und dieses wieder auf A. Seinen mathematischen Ausdruck findet ein Problem in den Formeln 3 und 4. Wie die Programme **XSIN** und **XCOS** beweisen, zeigen sich hier keine größeren Schwierigkeiten, man sollte sich zunächst nur überlegen, daß man zwei Programme braucht und sich entsprechende Namen ausdenken.

Genauso einfach programmiert man dann auch **SINX** und **COSX**, die algebraische Lösung der Formeln 3 und 4. Ein Problem bei den Rekursionsformeln ist, daß das Minuszeichen vor den Integralen in Formel 1, 2 und 4 zu Minusklammern führt, die COLCT nicht auflösen kann. Das Ergebnis der Berechnung bleibt daher sehr unübersichtlich und beschäftigt den Equation-writer erbärmlich lange.

Eine User-Defined-Function läßt sich in algebraische Ausdrücke integrieren, wie z.B. die Funktion Sinus, also statt 'SIN(X)' hieße es 'XEXP(X;3;1)'. Es müssen ja drei Elemente in der Klammer stehen, da das Programm XEXP mit der Definition von drei lokalen Variablen beginnt.

Bei den noch folgenden Programmen können nur die mit "(auch 28S)" gekennzeichneten unverändert auch auf diesem Rechner laufen.

Chaos schaffen ist nicht schwer, Ordnung machen dagegen sehr: Daher ein paar Sortierprogramme.

Zunächst zum Prinzip der folgenden Sortierprogramme am Beispiel von **SORTKERN** (auch für 28S): Dieses Programm erwartet in Ebene 1 eine Zahl n, welche nicht kleiner als drei ist (nur der ganzzahlige Anteil wird gebraucht) und angibt, daß die Ebenen 2 bis n+1 sortiert werden sollen. Diese Definition von n mutet etwas seltsam an, aber die eingebauten Funktionen ROLL, DROPN u.s.w. verwenden exakt die gleiche Art, die zu bearbeitenden Stackebenen zu quantifizieren.

Nun kommen wir zu den beiden wichtigsten Stellen im Programm SORTKERN, nämlich der TEST in Zeile 4 und die Entsorgung der sortierten Daten mit RAUS in Zeile 7. Zunächst zu TEST: Hier muß man sich etwas einfallen lassen. Üblicherweise beginnt man mit dem Duplizieren (Standardfunktion DUP2) der zu vergleichenden Daten, dann kommt eine der Testfunktionen, die im HP-48SX arbeiten wie z.B. die Grundrechenarten: Der Rechner verbraucht die Objekte in Ebene 1 und 2 um sie durch eine 0 oder 1 (die Testergebnisse) zu ersetzen. Damit ist TEST auch schon beendet. Nach dem Ende der START...NEXT-Schleife (Zeilen 3-7) enthält Ebene 1 ein sortiertes Element, welches gesichert werden muß, z.B. mit a ROLLD. Es kann hier auch der Name 'XYZ' (=beliebig) STO+ stehen, wenn XYZ eine Liste enthält, kommt nämlich jedes neue Element an den Anfang dieser Liste bzw. mit 'XYZ' SWAP STO+ an das Ende derselben.

Um n Objekte zu sortieren, führt SORTKERN n(n-1)/2 Tests durch, der Rechenaufwand steigt also quadratisch mit der Anzahl (n) der zu sortierenden Objekte. Wem die Details noch nicht ausreichen, um dieses Programmfragment in seinen eigenen Programmen einzusetzen, sollte die folgenden Beispiele beachten.

Hier zunächst **SORT**, dieses universelle Sortierprogramm erwartet folgende Eingaben:

1. Eine Liste in Ebene 1, die entweder nur reelle Zahlen oder nur Strings oder nur Binärzahlen oder nur Namen enthält, die allesamt die gleichen genannten Datentypen enthalten,
2. ein Array (Matrix oder Vektor) mit reellen Zahlen,
3. einen Namen, der eine Liste oder Array enthält,
4. eine positive Zahl n, die nicht kleiner als 3 ist und angibt, daß die Ebenen 2 bis n+1 sortiert werden sollen (zugelassene Elemente siehe 1.).

SORT schreibt das größte Element nach ganz rechts in Array/Liste, bzw. in Ebene 1 beim Sortieren des Stacks, das legt die Funktion "Größer" in Zeile 21 fest. Verwendet man hier "Kleiner", so dreht sich auch die entsprechende Reihenfolge um.

Zum Sortieren von Strings ist noch folgendes anzumerken: Das maßgebende Werk der (bundes-)deutschen Rechtschreibung, der Duden, sortiert Ä wie A, Ö wie O und Ü wie U. Die sprachlich nicht weniger mächtige Bundespost hingegen sortiert in ihren Telefonbüchern Ä wie AE, Ö wie OE und Ü wie UE. Der HP-48SX sortiert nach ASCII-Codes und bietet so eine dritte Variante: Ä, Ö und Ü werden nach Z eingeordnet. Außerdem unterscheiden die genannten Werke nicht zwischen Groß- und Kleinbuchstaben, der 48SX aber schon: "a" kommt hinter "Z" !!!

SORT kann sich bis zu zweimal selbst aufrufen, daher sollte man bei einer Namensänderung auch SORT in den Zeilen 10, 11 und 16 ändern.

MESO (MEnuSOrt) sortiert das aktuelle Directory alphabetisch. Zuerst erscheinen die in diesem Directory enthaltenen Subdirectorys und dann die Namen der übrigen Variablen. Damit MESO in jedem Directory funktioniert, sollte MESO und das benötigte Unterprogramm SORT im HOME-Directory stehen.

DSORT sortiert Daten als Ergänzung zu den Funktionen des TIME-Menues. Die Daten müssen dem eingestellten Modus

```
48SX →MJ
« { V -9 } DUP2
":Monat: " 1 DATE
FP %T IP + + INPUT
OBJ→ ROT ROT
":Jahr: " 1 DATE
%T FP 10000 * + +
INPUT OBJ→
» Bytes : 111
Checksum : # E4C7h
```

```
48SX FR13
« RCLF STD 0
"START ?" →MJ 10000
/ + 1 %
  DO FP 6 + DUP 0
TSTR
    IF DUP "F" POS
      THEN 2 "r  13"
REPL 1 12 SUB ROT 1
+ DUP 7 MOD 4 ROLLD
DISP 0
      END DROP 35
DATE+
    UNTIL KEY
  END DROP DROP2
STOF 3 FREEZE
» Bytes : 188,5
Checksum : # 29D8h
```

```
Fr  13.09.91
Fr  13.12.91
Fr  13.03.92
```

```
48SX KAL
« RCLF
"Druckbeginn im :"
→MJ DTAG DUP2 10000
/ + 1 % 1 + CJ {
"JANUAR" "FEBRUAR"
"MÄRZ" "APRIL"
"MAI" "JUNI" "JULI"
"AUGUST"
"SEPTEMBER"
"OKTOBER"
"NOVEMBER"
"DEZEMBER" } 1
"Anzahl der Monate
?"
":▓Monate: " INPUT
OBJ→
  START 4 ROLL GETI
"                 " 27
CHR + 253 CHR +
OVER CR SIZE 13 SUB
SWAP + CR PR1 DROP
4 ROLL "      " OVER
+ 27 CHR + 252 CHR
+ PR1 10 CHR
"  Mo Di Mi Do Fr S
a So"
+ 10 CHR + PR1
DROP2 -64 FS? + ROT
"        " 1 6 PICK
7 MOD SUB DUP DUP +
+ "" + 1 1 6 PICK
% 1 + 5 PICK
1000000 / + CJ DUP
5 ROLLD 8 ROLL -
    FOR d 9 d < " "
"  " IFTE + d +
      IF DUP SIZE
22 ≥
      THEN PR1 DROP
"  "
      END
    NEXT PR1
    IF SIZE 1 >
    THEN CR
    END
  NEXT 4 DROPN STOF
» Bytes : 701,5
Checksum : # 1D1Dh
```

```
     DEZEMBER
       1990

Mo Di Mi Do Fr Sa So
                 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
```

```
      JANUAR
       1991

Mo Di Mi Do Fr Sa So
    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
```

```
48SX MCOD
« ,1 SWAP
"▶▓∡▓≠^∫█→▓π∑∨▓≥
∂√α▓←▓▓¿¥K▓" i£§¯ê
óáP¹²µ±▓Ç▓▓¶ö»"
"ABCDEFGHIJKLMNOPQR
STUVWXYZ0123456789.
,-'?#*=/:ÄÖÜÄÉÑ"
ROT 1 OVER SIZE
  FOR n 3 DUPN n
DUP SUB DUP MEM NOT
    IF DISP POS
    THEN LASTARG
DUP SUB NUM 128 -
      DO 2000 OVER
2 MOD 3 1 IFTE 7
PICK * BEEP 2 / IP
DUP
        UNTIL 1 SAME
      END 2
    ELSE 5
    END 6 PICK *
WAIT DROP
  NEXT 4 DROPN
» Bytes : 302
Checksum : # 9415h
```

```
48SX STR
« 134 CHR + 145 CHR
+ 149 CHR + 137 CHR
+ 130 CHR + 148 CHR
+ 139 CHR + 144 CHR
+ 132 CHR + 158 CHR
+ 141 CHR + 146 CHR
+ 135 CHR + 133 CHR
+ 143 CHR + 150 CHR
+ 155 CHR + 138 CHR
+ 136 CHR + 131 CHR
+ 140 CHR + 152 CHR
+ 142 CHR + 153 CHR
+ 157 CHR + 147 CHR
+ 191 CHR + 190 CHR
+ 188 CHR + 184 CHR
+ 176 CHR + 160 CHR
+ 161 CHR + 163 CHR
+ 167 CHR + 175 CHR
+ 234 CHR + 243 CHR
+ 225 CHR + 222 CHR
+ 204 CHR + 170 CHR
+ 181 CHR + 177 CHR
+ 169 CHR + 199 CHR
+ 154 CHR + 151 CHR
+ 156 CHR + 182 CHR
+ 164 CHR + 187 CHR
+
» Bytes : 816
Checksum : # 9B1Fh
```

```
48SX SORTKERN
« 2
FOR a 2 a
  START
    IF TEST
    THEN SWAP
    END a ROLLD
  NEXT RAUS -1
STEP
» Bytes : 68,5
Checksum : # AFE3h
```

```
48SX SORT
«
  IF DUP TYPE
  THEN
    IFERR RCL
    THEN SIZE → s
      « LASTARG
        IFERR LIST→
        THEN ARRY→
LIST→ DUP2 DROPN *
SORT s →ARRY
        ELSE SORT s
→LIST
        END
      »
    ELSE LASTARG
SWAP SORT SWAP STO
    END
  ELSE 2
    FOR a 2 a
      START
        IF DUP2 >
        THEN SWAP
        END a ROLLD
      NEXT LASTARG
ROLLD -1
    STEP
  END
» Bytes : 209,5
Checksum : # 1340h
```

```
1: { "AAA" "AA" "Ä" "a"
"Z" "z" }

SORT

1: { "AA" "AAA" "Z" "a"
"z" "Ä" }
```

Ralf Pfeifer (116)

```
48SX MESO
« "{" VARS OBJ→ → n
  « 1 n
    START DUP VTYPE
15 SAME "" "'"
IFTE SWAP + n ROLL
    NEXT n SORT 1 n
    START +
    NEXT STR→ ORDER
  »
» Bytes : 121
Checksum : # 47DCh
```

```
48SX DSORT
«
  IF DUP TYPE
  THEN
    IFERR RCL
    THEN SIZE → s
      « LASTARG
        IFERR LIST→
        THEN ARRY→
LIST→ DUP2 DROPN *
DSORT s →ARRY
        ELSE DSORT
s →LIST
        END
      »
    ELSE LASTARG
SWAP DSORT SWAP STO
    END
  ELSE 2
    FOR a 2 a
      START
        IF DUP2
DDAYS ARG
        THEN SWAP
        END a ROLLD
      NEXT LASTARG
ROLLD -1
    STEP
  END
» Bytes : 215
Checksum : # 2580h
```

```
6: 2,031966
5: 1,032011
4: 6,061599
3: 3,111852
2: 4,061776
1: 5

DSORT

5: 6,061599
4: 4,061776
3: 3,111852
2: 2,031966
1: 1,032011
```

```
48SX ΣSORT
« ΣPAR 1 GET
"X-Col : " OVER + 1
DISP NΣ → x n
  « 1 NΣ
    START Σ-
    NEXT n 2
    FOR a 2 a
      START
        IF OVER x
GET OVER x GET <
        THEN SWAP
        END a ROLLD
      NEXT Σ+ -1
    STEP Σ+
  »
» Bytes : 151,5
Checksum : # C232h
```

```
ΣDAT
[[ -8 1 44 ]
 [ 7 2 5 ]
 [ 2,2 -8,6 9 ]
 [ ,23 5,1 -3 ]
 [ -12 7 -13 ]]

1 XCOL ΣSORT

[[ 7 2 5 ]
 [ 2,2 -8,6 9 ]
 [ ,23 5,1 -3 ]
 [ -8 1 44 ]
 [ -12 7 -13 ]]
```

```
48SX MKRY
« OVER SIZE SWAP
  DO DUP +
  UNTIL DUP2 SIZE <
  END 1 ROT SUB XOR
» Bytes : 47,5
Checksum : # E448h
```

```
2: "SICH LEGEN BRINGT SE
GEN"
1: "KINDERGELD"

MKRY

"▪▪▪e▪▪▪▪▪k▪▪▪▪▪▪e…▪▪▪▪
```

```
48SX CODE
« 1 CF KRYPT
» Bytes : 23,5
Checksum : # 2DADh
```

```
48SX DCD
« 1 SF KRYPT 1 CF
» Bytes : 28,5
Checksum : # 1F78h
```

```
48SX KRYPT
«
"ABCDEFGHIJKLMNOPQR
STUVWXYZÄÖÜ 0123456
789.,?"
SIZE LASTARG DUP +
→ a b
  « 1 OVER SIZE
    FOR n b OVER n
DUP SUB POS
      IF 1 FS?
      THEN a SWAP -
      END SWAP
    NEXT SIZE →ARRY
OVER SIZE 1 1 ROT
    FOR n GETI 4
ROLL n b DUP2 5
PICK ROT DUP SUB
POS 5 ROLL + DUP
SUB REPL ROT ROT
    NEXT DROP2
  »
» Bytes : 233,5
Checksum : # F807h
```

```
2: "ALLES NEU MACHT DER
MAI"
1: "DOSEMARIE"

CODE

1: "EÄ0J101NZ30THUUEMJVB
1FV"

2: "EÄ0J101NZ30THUUEMJVB
1FV"
1: "DOSEMARIE"

DCD

1: "ALLES NEU MACHT DER
MAI"
```

```
48SX MOND
« 7,021989 DATE
DDAYS TIME 15,23
HMS+ HMS→ 24 / +
7,38264723009 / 0
ACOS * SIN LASTARG
COS 1 - -50 * SWAP
SIGN * 0 RND
"Phase %" →TAG
» Bytes : 132
Checksum : # 7C56h
```

```
48SX WO?
« DATE ,01 DUP2 MOD
+ 1 + CJ 7 / CEIL 7
* SWAP CJ - -7 / IP
1 + "Woche " →TAG
» Bytes : 90
Checksum : # 4296h
```

```
(Datum: 2.10.90   13 h)
MOND
1: Phase %: 95

WO?
1: Woche : 40
```

```
48SX CJ
« 15,101582 SWAP
DDAYS 2299161 +
"JDN " →TAG
» Bytes : 50
Checksum : # E716h

3,101990  CJ
JDN : 2448168
```

```
48SX JC
« 2299161 -
15,101582 SWAP
DATE+
» Bytes : 38,5
Checksum : # 34D1h

2448168  JC
3,10199
```

```
48SX DOW
« 0 TSTR 1 2 SUB
"MoDiMiDoFrSaSo"
"MOTUWETHFRSASU"
ROT POS DUP 1 + SUB
» Bytes : 75,5
Checksum : # F235h

3,101990  DOW
1: "Mi"
```

(Monat/Tag/Jahr oder Tag.Monat.Jahr, siehe Flag -42) entsprechen und dürfen in Listen, reellen Arrays, in Namen die Listen oder Arrays enthalten oder im Stack (Ebene 1: Anzahl der Daten, Ebene 2 bis n+1 enthalten die Daten) gespeichert sein. Die TEST-Bedingung lautet hier DUP2 DDAYS ARG (Zeilen 21/22), wenn das späteste (=am weitesten in der Zukunft liegende) Datum am Ende des Arrays/Liste bzw. in Ebene 1 nach dem Stacksortieren stehen soll. Um diese Reihenfolge umzudrehen, ist als Test DUP2 DDAYS ARG NOT zu verwenden.

Und hier das nächste Programm zum Selberstricken (nicht abgedruckt): Wer gerne Uhrzeiten sortiert, verwendet als Test (in DSORT Zeilen 021/022, in SORTKERN Zeile 004) DUP2 HMS- ARG.

Als letztes Sortierprogramm sei hier noch **ΣSORT** vorgestellt (für 28S ist noch STR nach OVER in Zeile 2 einzufügen). Folgendes Problem ergibt sich: In der Statistikmatrix ΣDAT halten sich die Datenpunkte völlig ungeordnet auf. Es soll aber eine Tabelle gedruckt werden, und die ist ja nur schön, wenn man mit dem kleinsten x-Wert beginnt und mit dem größten endet. ΣSORT erfragt nun aus der Liste ΣPAR welches nun die Spalte der x-Werte ist (steht in ΣPAR an erster Stelle), zeigt die Nummer dieser Spalte an (weil man sonst vielleicht die Arbeitsweise des Programms vergißt), speichert diese in der lokalen Variablen x und sortiert dann die Statistikmatrix. Der Test ist dabei etwas aufwendiger, da die Zeilen von ΣDAT als Vektoren im Stack stehen. Mit OVER x GET holt das Programm dann die zu vergleichenden x-Werte. Die RAUS-Routine besteht dagegen nur aus Σ+. Wenn ΣSORT fertig ist, kann mit Σ- Datenpunkt für Datenpunkt ordentlich sortiert aus ΣDAT geholt werden.

In PRISMA 90.01.43 habe ich für den HP-28S Programme zum Verschlüsseln und Morsen von Texten vorgestellt (Theorie siehe dort), hier nun die HP-48SX-Versionen: **CODE** erwartet in Ebene 2 den Klartext der mit Hilfe des Schlüsselwortes in Ebene 1 codiert werden soll, **DCD** erwartet in Ebene 2 den codierten Text, den es mit dem Schlüssel in Ebene 1 in Klartext verwandelt. **KRYPT** ist das Unterprogramm, welches die ganze Arbeit macht.

Durch die neue REPL-Funktion im 48SX sowie logische Verbesserungen ließen sich Geschwindigkeit und Platzbedarf von KRYPT deutlich optimieren. Der String in den Zeilen 2-4 definiert die Zeichen, die als Eingabe (Schlüssel, Klartext, Code) erlaubt sind und auf die sich das Programm bei der Ausgabe beschränkt. Diesen String darf man um beliebige Zeichen (z.B. Kleinbuchstaben) erweitern oder kürzen, das Hin- und Rückübersetzen klappt allerdings nur, wenn man jeweils den gleichen String verwendet. Da der Hp-48SX aber den Austausch von Daten ermöglicht (Module, IR-Diode), ist die von KRYPT bewerkstelligte Einschränkung des Zeichenvorrats oft überflüssig.

MKRY (auch 28S) ist eine unverschämt kurze und schnelle Version von KRYPT, wobei Hin- und Rückübersetzung das gleiche Programm erledigt. MKRY verknüpft die Zeichen mit XOR. Als Nachteil ergibt sich, daß dabei alle 256 Zeichen als Produkt von MKRY herauskommen können, z.B. die Null oder das String-Zeichen (''), wobei dann dann natürlich die EDIT-Funktion versagt, oder Steuerzeichen des Druckers, die einen vernüftigen Ausdruck verhindern. Die Bedienung ist einfach: Ebene 1 enthält den Schlüssel, Ebene 2 den Klartext oder den Code.

Auch das Morsen hat der 48SX gelernt. **MCOD** verwandelt Klartext, der als String (nur mit Großbuchstaben) inEbene 1 steht, in Töne, auch wenn sich mein 48er etwas erkältet anhört (der programmierte Jeo-Cocker-Sound). Die übersetzbaren Zeichen definiert der String in den Zeilen 5-7, dabei bedeutet das Multiplikationszeichen (*) Anfangszeichen, das Doppelkreuz (#) Schlußzeichen, btx-Benutzer kennen das. Der String in den Zeilen 2-4 enthält die entsprechenden Morsecodes. Um diesen einzugeben, tastet man das Programm STR ein, kopiert es (DUP oder ENTER) in Ebene 2, führt BYTES aus und vergleicht Prüfsumme & Bytes mit denen im Listing. Wenn sie stimmen, startet man mit EVAL. Dabei vereinfacht sich die Eingabe von STR, wenn man dem CST-Menue den String " CHR + " zuweist, dieser wird auf Tastendruck ohne " ins Programm eingefügt. Jetzt beginnt man mit der Eingabe von MCOD und holt den String mit dem interaktiven Stack (zu starten mit ↑STK im EDIT-Menue, dann ECHO ENTER) ins Programm. Die ,1 zu Beginn von MCOD bestimmt, wie lange Punkt und Strich piepsen und die Zahl in Zeile 14 legt die Frequenz der Töne fest.

Während MCOD läuft, zeigt es den gerade gepiepsten Buchstaben an. Unbekannte Zeichen und SPACE ergeben Pausen.

Von Zeit zu Zeit räumt der 48SX sein RAM auf. Dabei tritt eine kurze Lähmung ein, die beim Morsen vom Hörer als Pause mißverstanden werden kann. Dieses Aufräumen erzwingt das Programm daher durch den Befehl MEM während der Pausen zwischen den Buchstaben.

Die folgenden beiden Programme brauchen als Unterprogramm **→MJ**, welches nach Monat und Jahr fragt, und zugleich die aktuellen Werte aus dem Kalender anbietet. →MJ wurde so bereits in PRISMA 90.03.28 abgedruckt, dort als Unterprogramm zu DKAL.

KAL druckt den Kalender aus. Zuerst fragt das Programm nach dem Monat und Jahr des ersten zu druckenden Kalendermonats, dann nach der Anzahl der insgesamt zu druckenden Monate. Die Liste in den Zeilen 5-12 enthält die Monatsnamen, welche beliebig geändert werden können (z.B. englische Namen), sofern sie aus nicht mehr als 11 Zeichen bestehen. Das Schachbrett in Zeile 15 soll ein Dreieck sein, und 79,5 Bytes kann man auf folgende Weise sparen: 27 CHR + 253 CHR + in Zeilen 18/19 an den String anhängen, durch einen String mit zwei Zeichen die Befehle 27 CHR + 252 CHR ersetzen und den String in Zeilen 25/26 mit einem Newline-Befehl (blaue Vortaste, dann Komma) beginnen und beenden, dafür 10 CHR in Zeile 24 und + 10 CHR + in Zeile 27 streichen.

FR13 fragt zuerst nach dem Monat, ab dem nach einem Freitag den 13. gesucht werden soll und zeigt dann alle "gefährdeten" Daten an. Um FR13 abzubrechen, drückt man eine beliebige Taste außer ON.

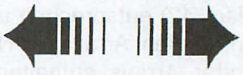
WO? ermittelt die Nummer der Woche im Jahr. Manche Produkte z.B. die Süßwaren von Ferrero, tragen auf der Packung die Produktionswoche. WO? berechnet aus dem eingestellten Datum die laufende Nummer der aktuellen Woche.

CJ berechnet das Julianische Datum (eine fortlaufende Tagesnumerierung) zu dem Datum in Ebene 1, **JC** kehrt das um und findet zum Julianischen Datum das normale Datum und **DOW** berechnet zu dem Datum in Ebene 1 den Wochentag.

MOND ist zugegebenermaßen ziemlich primitiv. Es geht davon aus, daß zwischen zwei Neumonden immer die gleiche Zeitspanne vergeht. Das Programm berechnet Neumond und Vollmond recht genau, schludert aber beim ersten und letzten Viertel (Halbmond) gewaltig, bis zu 8%. Da sich aber bei Halbmond die Phasen ohnehin schnell ändern, irrt sich das Programm um etwa einen Tag, außerdem, wer mißt schon die beleuchtete Fläche des Mondes genau aus. Eine positive Prozentzahl bedeutet zunehmenden Mond (=nach Neumond, am Abendhimmel), negative Prozente bedeuten abnehmender Mond (=nach Vollmond, am Morgenhimmel). 100% ist Vollmond, 0% Neumond. Im übrigen berechnet das Programm die Phase aus der aktuellen Uhrzeit/Datum. Die ermittelten Werte sind auf dem gesamten Globus ohne Längen oder Zeitkorrektur gültig!

Ralf Pfeiffer
Rubensstraße 5
500 Köln 50

Input-Output Board



zum Zweiten

HP41, IL-Modul, Extended I/O, CCD Modul, HP-IL Converter HP82166A

Hardware

Die Miniaturisierung in der Elektronik ist unaufhaltsam, das im letzten PRISMA vorgestellte Board wurde kurzerhand mitsamt seinen Steckkarten auf eine Europakarte gebracht, man siehe dazu den erweiterten Schaltplan.
Benötigt man einzelne Teile des Systems nicht, so kann man sie einfach unbestückt lassen. Dies kann natürlich nicht für die Steuerlogik gelten, diese wird für alle Teile benötigt.
Die Beschreibungen der Teile des Boards befinden sich alle im Heft PRISMA 3/90. Das einzige, was dort noch nicht aufgetaucht war, war die Ausnutzung der beiden freien NOR-Gatter.

Interrupt-Logik

Für Besitzer eines IL-Development Moduls ist es damit möglich den Rechner per IL-Converter wieder aufzuwecken, wenn dieser sich aus Langeweile ausgeschaltet hat.
Ein High an einem der vier MSQR-Eingänge erzeugt auf dem Pin MSQR des IL-Converters das aktive Low, das dieser zum Auslösen der IL-Meldung für den HP41 benötigt. Voraussetzung ist natürlich, daß das Input-Output Board mit Strom versorgt wird.
Besitzt man kein IL-Development Modul, so kann man mit dem Extended I/O-Modul zumindestens mittels einer Endlosschleife, in der der Converter permanent nach einem Service Request gefragt wird, eine externe Aktion an den Rechner weitergeben.

Stromversorgung

Die Platine benötigt normalerweise nur +5V, 500mA sollten genügen, solange man nicht allzu große Lasten damit betreiben will, Relais benötigen schon einige Hundert Milliampere pro Stück.
Die ±15V sind nur für die D/A und A/D-Wandler nötig, werden diese nicht bestückt, so kann man sich diese Spannungsversorgung sparen. Hier genügen 200-300 mA pro Spannung, hier werden ja keine großen Lasten getrieben.

IL-Development Modul und I/O Board

Dieses Programmpaket ist für die Benutzung von IL-Development-Modul und dem HP82166A I/O Board verfaßt. Die Funktionen sind äquivalent zu den Funktionen, die bei den Programmen im letzten Artikel im PRISMA 3/90 verwendet worden waren, in denen das X-Function Modul Verwendung gefunden hatte.
Die Unterscheidung der Programme erfolgt durch ein vorangestelltes *"D"*.

Eine Erweiterung stellt die Funktion ***"DMSRQ"*** dar:
Bei eingeschaltetem Rechner und nicht laufenden Programmen kann nach Ausführung dieses Programmes ein "Manual Service Request" vom HP41 erkannt werden; dieser springt die dafür vorgesehene Unterroutine *"INTR"* an, hier können beliebige Aktionen ausgeführt werden.
Es ist also somit möglich den HP41 über die Leitungen MSRQ I-IV von außen anzuwerfen, sich z.B. durch einen BEEP bemerkbar zu machen.

DIN/OUT	→	setzt HP82166A Converterstatus für das I/O-Board
DOUT1	→	schreibt Byte aus dem X-Register auf das Output Modul 1
DOUT2	→	schreibt Byte aus dem X-Register auf das Output Modul 2
DIN1	→	Liest ein Byte vom Input-Modul 1 in das X-Register
DIN2	→	Liest ein Byte vom Input-Modul 2 in das X-Register
DMSRQ	→	Aktiviert die Service-Request-Routine *"INTR"* für ein aktives Signal an MSRQ I-IV

Stückliste

Anzahl	Nr.	Bezeichnung
001	IC1	74123
003	IC2	1/4 7400
004	IC3	1/4 7402
004	IC4	1/4 7402
001	IC5	ADC0804
001	IC6	TL072
001	IC7	74HC574
001	IC8	DAC0808
002	IC9	TL072
001	IC10	74HC574
001	IC11	74HC574
001	IC12	74HC574
001	R1	470k
001	R2	120k
001	R3	10k
001	R4	100R
001	R5	7k5
001	R6	10k
001	R7	10k
001	R8	1K
001	R9	2k
001	R10	5k0
001	R11	10k
001	R12	10k
001	R13	10k
001	R14	10k
001	R15	10k
001	C1	220n
001	C2	220n
001	C3	150p
001	C4	1u
001	C5	68p
001	C6	15p
001	Z1	LM336
001	D1	1N4148
001	D2	1N4148

```
01 LBL "DIN/DOUT"
02 SF 33
03 19
04 BSIZEX
05 1
06 TAD
07 0
08 DDT
09 0
10 PT=
11 19
12 INBUFX
13 UNT
14 0
15 PT=
16 0
17 X - BUF
18 2
19 PT=
20 0
21 X - BUF
22 2
23 PT=
24 16
25 X - BUF
26 3
27 PT=
28 0
29 X - BUF
30 1
31 LAD
32 0
33 DDL
34 0
35 PT=
36 19
37 OUTBUFX
```

```
01 LBL "DOUT1"
02 SF 33
03 0
04 PT=
05 MIPT
06 RDN
07 X - BUF
08 1
09 LAD
10 SDC
11 OUTBUFX
12 AIPT
13 CF 33
14 RDN
15 END
```

```
01 LBL "DOUT2"
02 SF 33
03 0
04 PT=
05 MIPT
06 RDN
07 X - BUF
08 1
09 LAD
10 LPD
11OUTBUFX
12 AIPT
13 CF 33
14 RDN
15 END
```

```
38 UNL
39 CF 33
40 CLX
41 END
```

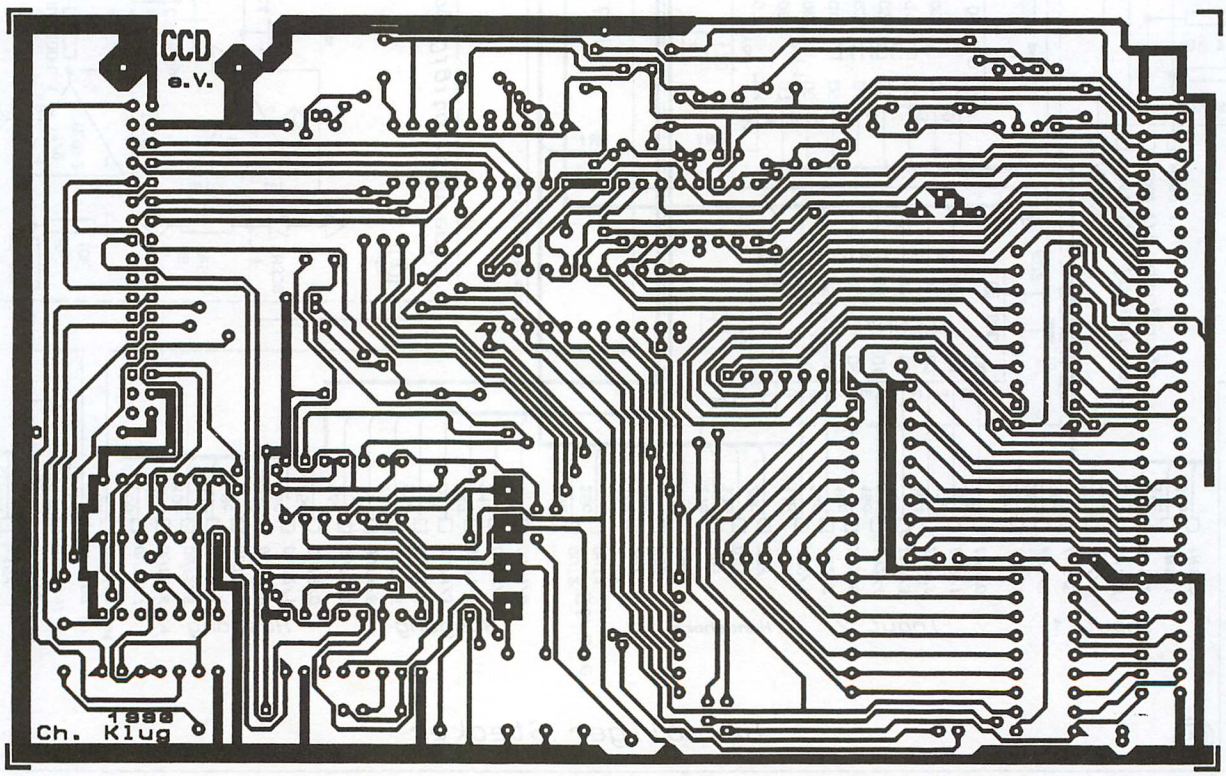
```
01 LBL "DIN1"
02 SF 33
03 1
04 LAD
05 SDC
06 CF 33
07 TRIGGER
08 SF 33
09 0
10 PT=
11 MIPT
12 1
13 TAD
14 INBUFX
15 BUF - XB
16 AIPT
17 CF 33
18 END
```

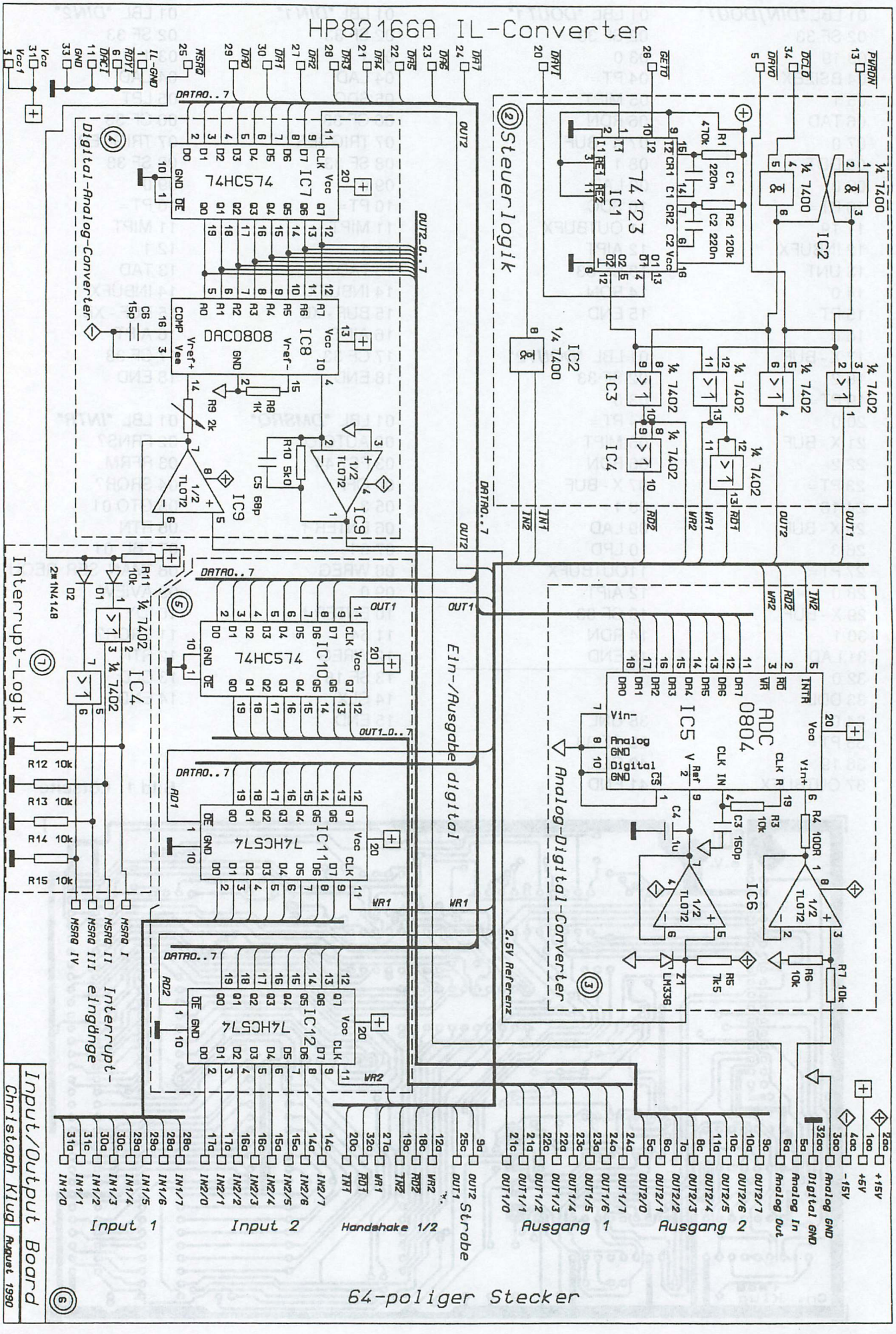
```
01 LBL "DMSRQ"
02 AUTOIO
03 FS? 44
04 OFF
05 3
06 ENTER ↑
07 64
08 WREG
09 0
10 ENTER ↑
11 64
12 WREG
13 SF 18
14 CLX
15 END
```

```
01 LBL "DIN2"
02 SF 33
03 1
04 LAD
05 LPT
06 CF 33
07 TRIGGER
08 SF 33
09 0
10 PT=
11 MIPT
12 1
13 TAD
14 INBUFX
15 BUF - XB
16 AIPT
17 CF 33
18 END
```

```
01 LBL "INTR"
02 FRNS?
03 RFRM
04 SRQR?
05 GTO 01
06 RTN
07 LBL 01
08 "MAN SER REQU"
09 AVIEW
10 IDY
11 SRQR?
12 RTN
13 CLD
14 END
```

Bild 1: Lötseite




HP82166A IL-Converter
Steuerlogik
Digital-Analog-Converter
Ein-/Ausgabe digital
Analog-Digital-Converter
Interrupt-Logik
Interrupt-eingänge
Input/Output Board
Christoph Klug
August 1990
Input 1
Input 2
Handshake 1/2
Ausgang 1
Ausgang 2
64-poliger Stecker

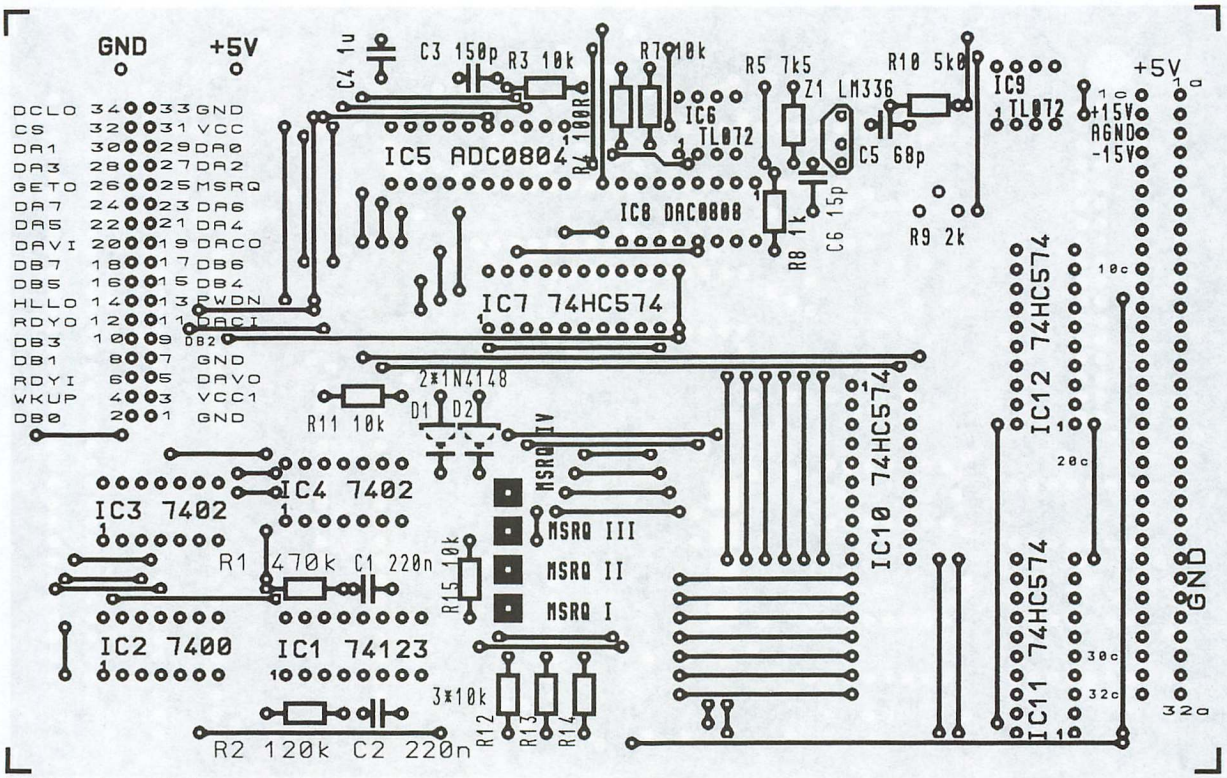


Bild 2: Bestückungsseite

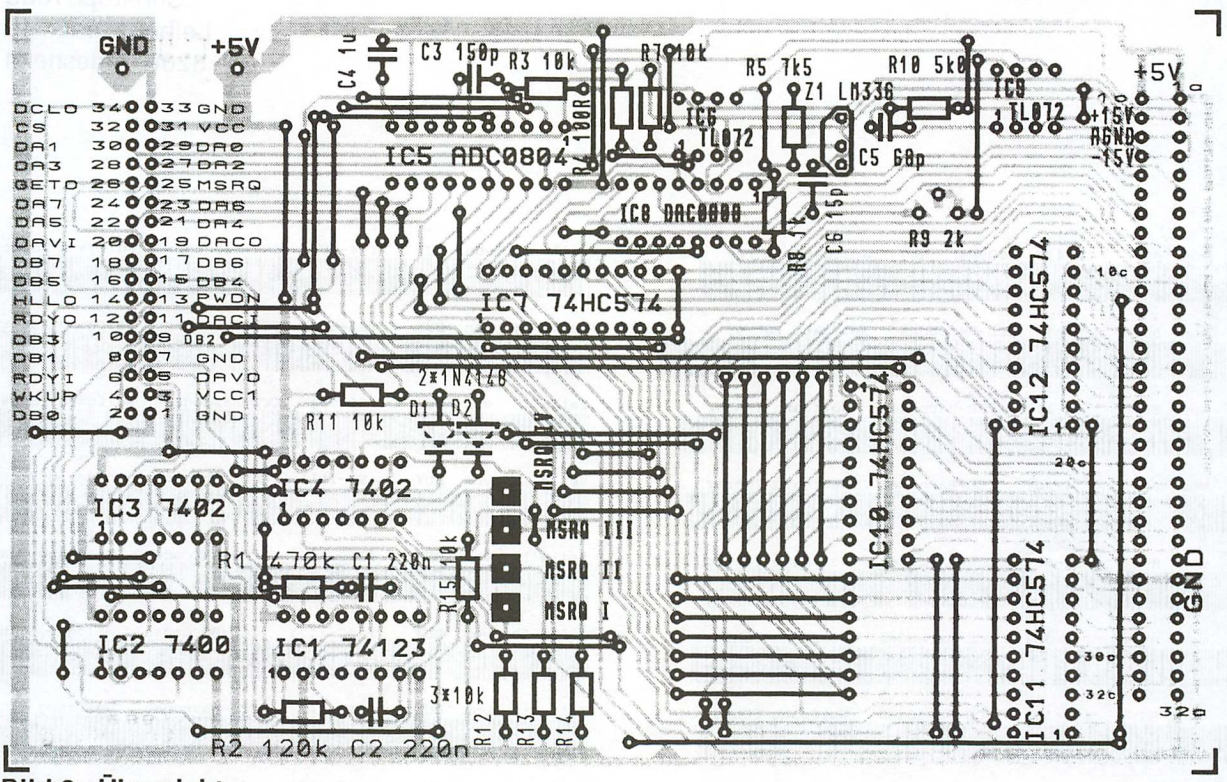


Bild 3: Übersicht

So schön wie das letzte Layout ist dieses leider nicht geworden, die Busstrukturen führten zu vielen Brükken. Die letzten Leiterbahnen fanden nur noch so ihren Weg zum Stecker. Die vier Eingänge für die Interrupts habe ich leider nicht mehr zum Stecker führen können, sie werden auch wohl eher seltenere Anwendung finden.

Sonst sind alle Signale auf dem 64-poligen Stecker zu finden, die Pinbelegung bitte dem Schaltplan entnehmen, dafür ist dieser ja da. Die extra Pfosten für die +5V-Versorgung können weitere Platinen versorgen, die eigentliche Stromversorgung sollte über die 64-polige Steckerleiste erfolgen.

Bitte unbedingt auf die etwas unregelmäßige Ausrichtung der ICs achten, es ließ sich leider nicht vermeiden, es ging schließlich darum, so wenig wie möglich Brücken zu produzieren.

Barcodes gibt's leider keine, die paar Zeilen kann man aber getrost eintippen, Übung schadet nicht.

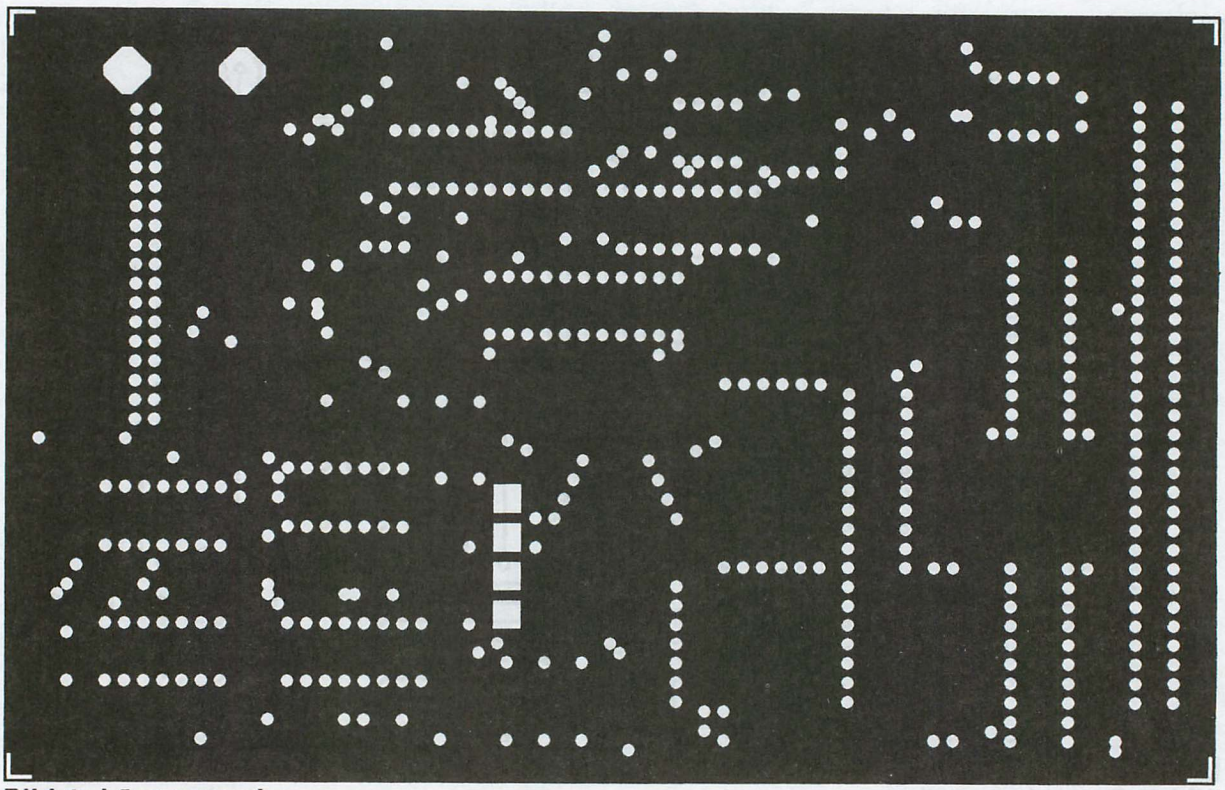


Bild 4: Lötstopmaske

Christoph Klug
Leibnitzstraße 19
3200 Hildesheim

BUY

Zeile 1 von BUY (1-4) CCD-Barcodes Wolfgang Führer

Zeile 2 von BUY (5-9) CCD-Barcodes Wolfgang Führer

Zeile 3 von BUY (10-13) CCD-Barcodes Wolfgang Führer

Zeile 4 von BUY (14-20) CCD-Barcodes Wolfgang Führer

Zeile 5 von BUY (21-23) CCD-Barcodes Wolfgang Führer

Zeile 6 von BUY (24-30) CCD-Barcodes Wolfgang Führer

Zeile 7 von BUY (31-39) CCD-Barcodes Wolfgang Führer

Zeile 8 von BUY (40-46) CCD-Barcodes Wolfgang Führer

Zeile 9 von BUY (47-53) CCD-Barcodes Wolfgang Führer

Zeile 10 von BUY (54-58) CCD-Barcodes Wolfgang Führer

Zeile 11 von BUY (59-66) CCD-Barcodes Wolfgang Führer

Zeile 12 von BUY (67-73) CCD-Barcodes Wolfgang Führer

Zeile 13 von BUY (74-80) CCD-Barcodes Wolfgang Führer

Zeile 14 von BUY (81-88) CCD-Barcodes Wolfgang Führer

Zeile 15 von BUY (89-96) CCD-Barcodes Wolfgang Führer

Zeile 16 von BUY (97-101) CCD-Barcodes Wolfgang Führer

Zeile 17 von BUY (102-106) CCD-Barcodes Wolfgang Führer

Zeile 18 von BUY (107-112) CCD-Barcodes Wolfgang Führer

Zeile 19 von BUY (113-117) CCD-Barcodes Wolfgang Führer

Zeile 20 von BUY (118-124) CCD-Barcodes Wolfgang Führer

Zeile 21 von BUY (125-132) CCD-Barcodes Wolfgang Führer

Zeile 22 von BUY (133-139) CCD-Barcodes Wolfgang Führer

Zeile 23 von BUY (140-145) CCD-Barcodes Wolfgang Führer

Zeile 24 von BUY (146-148) CCD-Barcodes Wolfgang Führer

KEYS


Zeile 1 von KEYS (1-2) CCD-Barcodes Dr.M. Hochenegger


Zeile 2 von KEYS (3-5) CCD-Barcodes Dr.M. Hochenegger


Zeile 3 von KEYS (6-6) CCD-Barcodes Dr.M. Hochenegger


Zeile 4 von KEYS (7-8) CCD-Barcodes Dr.M. Hochenegger


Zeile 5 von KEYS (9-10) CCD-Barcodes Dr.M. Hochenegger


Zeile 6 von KEYS (11-11) CCD-Barcodes Dr.M. Hochenegger


Zeile 7 von KEYS (12-13) CCD-Barcodes Dr.M. Hochenegger


Zeile 8 von KEYS (14-14) CCD-Barcodes Dr.M. Hochenegger


Zeile 9 von KEYS (15-16) CCD-Barcodes Dr.M. Hochenegger


Zeile 10 von KEYS (17-17) CCD-Barcodes Dr.M. Hochenegger


Zeile 11 von KEYS (18-19) CCD-Barcodes Dr.M. Hochenegger


Zeile 12 von KEYS (20-20) CCD-Barcodes Dr.M. Hochenegger


Zeile 13 von KEYS (21-22) CCD-Barcodes Dr.M. Hochenegger


Zeile 14 von KEYS (23-23) CCD-Barcodes Dr.M. Hochenegger


Zeile 15 von KEYS (24-25) CCD-Barcodes Dr.M. Hochenegger


Zeile 16 von KEYS (26-26) CCD-Barcodes Dr.M. Hochenegger


Zeile 17 von KEYS (27-28) CCD-Barcodes Dr.M. Hochenegger


Zeile 18 von KEYS (29-29) CCD-Barcodes Dr.M. Hochenegger


Zeile 19 von KEYS (30-30) CCD-Barcodes Dr.M. Hochenegger


Zeile 20 von KEYS (31-31) CCD-Barcodes Dr.M. Hochenegger


Zeile 21 von KEYS (32-32) CCD-Barcodes Dr.M. Hochenegger


Zeile 22 von KEYS (33-34) CCD-Barcodes Dr.M. Hochenegger


Zeile 23 von KEYS (35-35) CCD-Barcodes Dr.M. Hochenegger


Zeile 24 von KEYS (36-37) CCD-Barcodes Dr.M. Hochenegger


Zeile 25 von KEYS (38-38) CCD-Barcodes Dr.M. Hochenegger


Zeile 26 von KEYS (39-40) CCD-Barcodes Dr.M. Hochenegger


Zeile 27 von KEYS (41-41) CCD-Barcodes Dr.M. Hochenegger


Zeile 28 von KEYS (42-43) CCD-Barcodes Dr.M. Hochenegger


Zeile 29 von KEYS (44-44) CCD-Barcodes Dr.M. Hochenegger


Zeile 30 von KEYS (45-46) CCD-Barcodes Dr.M. Hochenegger


Zeile 31 von KEYS (47-47) CCD-Barcodes Dr.M. Hochenegger


Zeile 32 von KEYS (48-49) CCD-Barcodes Dr.M. Hochenegger


Zeile 33 von KEYS (50-50) CCD-Barcodes Dr.M. Hochenegger


Zeile 34 von KEYS (51-51) CCD-Barcodes Dr.M. Hochenegger


Zeile 35 von KEYS (52-52) CCD-Barcodes Dr.M. Hochenegger


Zeile 36 von KEYS (53-53) CCD-Barcodes Dr.M. Hochenegger


Zeile 37 von KEYS (54-55) CCD-Barcodes Dr.M. Hochenegger


Zeile 38 von KEYS (56-56) CCD-Barcodes Dr.M. Hochenegger


Zeile 39 von KEYS (57-58) CCD-Barcodes Dr.M. Hochenegger


Zeile 40 von KEYS (59-59) CCD-Barcodes Dr.M. Hochenegger

Zeile 41 von KEYS (60-61) CCD-Barcodes Dr.M. Hochenegger

Zeile 42 von KEYS (62-62) CCD-Barcodes Dr.M. Hochenegger

Zeile 43 von KEYS (63-64) CCD-Barcodes Dr.M. Hochenegger

Zeile 44 von KEYS (65-65) CCD-Barcodes Dr.M. Hochenegger

Zeile 45 von KEYS (66-67) CCD-Barcodes Dr.M. Hochenegger

Zeile 46 von KEYS (68-68) CCD-Barcodes Dr.M. Hochenegger

Zeile 47 von KEYS (69-69) CCD-Barcodes Dr.M. Hochenegger

Zeile 48 von KEYS (70-70) CCD-Barcodes Dr.M. Hochenegger

Zeile 49 von KEYS (71-71) CCD-Barcodes Dr.M. Hochenegger

Zeile 50 von KEYS (72-73) CCD-Barcodes Dr.M. Hochenegger

Zeile 51 von KEYS (74-74) CCD-Barcodes Dr.M. Hochenegger


Zeile 52 von KEYS (75-76) CCD-Barcodes Dr.M. Hochenegger


Zeile 53 von KEYS (77-77) CCD-Barcodes Dr.M. Hochenegger


Zeile 54 von KEYS (78-79) CCD-Barcodes Dr.M. Hochenegger


Zeile 55 von KEYS (80-80) CCD-Barcodes Dr.M. Hochenegger


Zeile 56 von KEYS (81-82) CCD-Barcodes Dr.M. Hochenegger


Zeile 57 von KEYS (83-83) CCD-Barcodes Dr.M. Hochenegger


Zeile 58 von KEYS (84-85) CCD-Barcodes Dr.M. Hochenegger


Zeile 59 von KEYS (86-86) CCD-Barcodes Dr.M. Hochenegger


Zeile 60 von KEYS (87-87) CCD-Barcodes Dr.M. Hochenegger


Zeile 61 von KEYS (88-88) CCD-Barcodes Dr.M. Hochenegger


Zeile 62 von KEYS (89-90) CCD-Barcodes Dr.M. Hochenegger


Zeile 63 von KEYS (91-91) CCD-Barcodes Dr.M. Hochenegger


Zeile 64 von KEYS (92-92) CCD-Barcodes Dr.M. Hochenegger


Zeile 65 von KEYS (93-94) CCD-Barcodes Dr.M. Hochenegger


Zeile 66 von KEYS (95-95) CCD-Barcodes Dr.M. Hochenegger


Zeile 67 von KEYS (96-97) CCD-Barcodes Dr.M. Hochenegger


Zeile 68 von KEYS (98-98) CCD-Barcodes Dr.M. Hochenegger


Zeile 69 von KEYS (99-100) CCD-Barcodes Dr.M. Hochenegger


Zeile 70 von KEYS (101-101) CCD-Barcodes Dr.M. Hochenegger


Zeile 71 von KEYS (102-103) CCD-Barcodes Dr.M. Hochenegger


Zeile 72 von KEYS (104-104) CCD-Barcodes Dr.M. Hochenegger


Zeile 73 von KEYS (105-106) CCD-Barcodes Dr.M. Hochenegger


Zeile 74 von KEYS (107-107) CCD-Barcodes Dr.M. Hochenegger

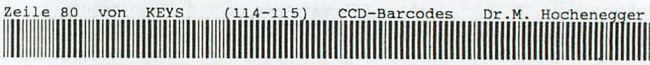

Zeile 75 von KEYS (108-108) CCD-Barcodes Dr.M. Hochenegger


Zeile 76 von KEYS (109-109) CCD-Barcodes Dr.M. Hochenegger


Zeile 77 von KEYS (110-110) CCD-Barcodes Dr.M. Hochenegger


Zeile 78 von KEYS (111-112) CCD-Barcodes Dr.M. Hochenegger


Zeile 79 von KEYS (113-113) CCD-Barcodes Dr.M. Hochenegger


Zeile 80 von KEYS (114-115) CCD-Barcodes Dr.M. Hochenegger


Zeile 81 von KEYS (116-116) CCD-Barcodes Dr.M. Hochenegger


Zeile 82 von KEYS (117-118) CCD-Barcodes Dr.M. Hochenegger


Zeile 83 von KEYS (119-119) CCD-Barcodes Dr.M. Hochenegger

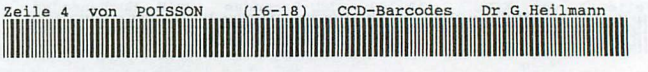

Zeile 84 von KEYS (120-121) CCD-Barcodes Dr.M. Hochenegger

POISSON


Zeile 1 von POISSON (1-3) CCD-Barcodes Dr.G.Heilmann


Zeile 2 von POISSON (4-10) CCD-Barcodes Dr.G.Heilmann


Zeile 3 von POISSON (11-15) CCD-Barcodes Dr.G.Heilmann


Zeile 4 von POISSON (16-18) CCD-Barcodes Dr.G.Heilmann


Zeile 5 von POISSON (19-22) CCD-Barcodes Dr.G.Heilmann


Zeile 6 von POISSON (23-27) CCD-Barcodes Dr.G.Heilmann


Zeile 7 von POISSON (28-34) CCD-Barcodes Dr.G.Heilmann

Zeile 8 von POISSON (35-43) CCD-Barcodes Dr.G.Heilmann

Zeile 9 von POISSON (44-48) CCD-Barcodes Dr.G.Heilmann

Zeile 10 von POISSON (49-56) CCD-Barcodes Dr.G.Heilmann

Zeile 11 von POISSON (57-59) CCD-Barcodes Dr.G.Heilmann

ZT

Zeile 1 von ZT (1-5) CCD-Barcodes Dr.G.Heilmann

Zeile 2 von ZT (6-13) CCD-Barcodes Dr.G.Heilmann

Zeile 3 von ZT (14-17) CCD-Barcodes Dr.G.Heilmann

Zeile 4 von ZT (18-24) CCD-Barcodes Dr.G.Heilmann

Zeile 5 von ZT (25-34) CCD-Barcodes Dr.G.Heilmann

Zeile 6 von ZT (35-42) CCD-Barcodes Dr.G.Heilmann

Zeile 7 von ZT (43-44) CCD-Barcodes

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

CP/M-80
Peter-C. Spaeth, Michaeliburgstraße 4, 8000 München 80

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 Germann (3423), Hobeuken 18, 4322 Spockhövel 2,
✆ (02339) 39 63

Serie 80
Klaus Kaiser (1661), Mainzer Landstr. 561, 6230 Frankfurt 80,
✆ (069) 39 78 52

Vermessungswesen
Ulrich Kulle (2719), Memeler Straße 26, 3000 Hannover 51,
✆ (0511) 60 42 72 8

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

Postvertriebsstück
Gebühr bezahlt

D 2856 F

CCD - Computerclub Deutschland e.V.
Schwalbacher Straße 50
D-6000 Frankfurt am Main 1

CCD
ISSN 0176-8735
PRISMA
Juli / August 1990 Nr. 4